PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :
G06K 9/00

A1

(11) International Publication Number: WO 94/09447

(43) International Publication Date: 28 April 1994 (28.04.94)

(21) International Application Number: PCT/US92/08703

(22) International Filing Date: 13 October 1992 (13.10.92)

(71)(72) Applicants and Inventors: ORBACH, Zvi [IL/US]; 331 Creekside, Palo Alto, CA 94306 (US). BARON, Ehud [IL/IL]; 63 Haag Street, 34 980 Haifa (IL).

(74) Agents: GALLOWAY, Peter, D. et al.; Ladas & Parry, 26 West 61 Street, New York, NY 10023 (US).

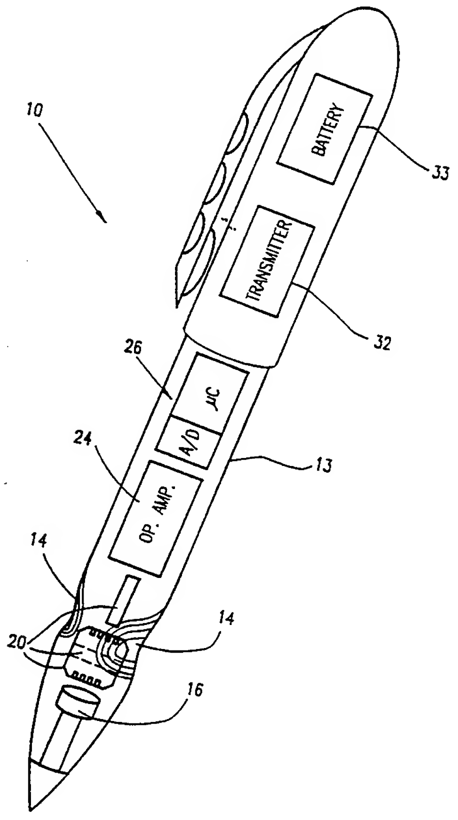
(81) Designated States: AT, AU, BB, BG, BR, CA, CH, CS, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, MG, MN, MW, NL, NO, PL, RO, RU, SD, SE, UA, US, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, SN, TD, TG).

Published
With international search report.

(54) Title: APPARATUS FOR READING HANDWRITING

(57) Abstract

Apparatus for reading handwriting including apparatus (20) for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written and apparatus, which is configured for the individual, for providing a non-individual dependent output (26) indicating the symbols being written in response to the sensed features.



APPARATUS FOR READING HANDWRITING

FIELD OF THE INVENTION

The present invention relates to computer input devices generally and more particularly to handwriting responsive computer input devices.

BACKGROUND OF THE INVENTION

There exists a significant amount of activity in the field of on-line handwriting recognition. The prior art current to 1990 is reviewed in "The State of the Art in On-Line Handwriting Recognition" by Charles C. Tappert et al, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August, 1990.

Generally speaking, on-line handwriting analysis is currently employed for two distinct applications: identity verification and input of handwritten letters and numbers into a computer. These two applications have sharply contrasting operational requirements and goals. Handwriting analysis for identity verification senses features of handwriting which are distinct for each individual and thus can be used to unambiguously identify a given individual. In contrast, handwriting analysis for alphanumeric input to a computer seeks to minimize the effect of the very features which are important for identity verification and to concentrate on universal handwriting characteristics which can be associated with given symbols independently of the individual writer.

Currently existing and proposed systems providing handwriting analysis for alphanumeric input to a computer are generally geared towards recognition of how a symbol looks rather than how it is created. Accordingly, such systems employ digitizers or graphic tablets.

Signature verification systems, on the other hand, attempt to identify biometric characteristics of the writer and employ indications such as pressure and acceleration during writing.

U.S. Patent 4,345,239 employs pen acceleration for use in a signature verification system. U.S. Patent 5,054,088 employs both acceleration and pressure data characteristics of handwriting for identity verification. As indicated by the above patents, pen acceleration is employed for signature verification because it is a personal feature, characteristic of each individual. Accordingly, pen acceleration has not been employed for alphanumeric input.

U.S. Patent 4,751,741 describes pen-type character recognition apparatus which employs pen pressure data exclusively.

SUMMARY OF THE INVENTION

The present invention seeks to provide improved handwriting information input apparatus.

There is thus provided in accordance with a preferred embodiment of the present invention apparatus for reading handwriting including apparatus for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written and apparatus, which is configured for the individual, for providing a non-individual dependent output indicating the symbols being written in response to the sensed features.

In accordance with a preferred embodiment of the present invention the apparatus for reading handwriting is contained in a hand-held housing. Preferably the apparatus for reading handwriting includes apparatus for wireless communication with a computer to which it inputs symbol data.

In accordance with a preferred embodiment of the present invention, the apparatus for reading handwriting does not require a tablet. Additionally in accordance with a preferred embodiment of the invention, the apparatus for reading handwriting communicates with the keyboard input of a computer.

Additionally in accordance with a preferred embodiment of the present invention there is provided apparatus for reading handwriting including personalized hand-held apparatus for sensing acceleration during handwriting and providing an output indication of handwriting content in a non-personalized form.

Further in accordance with a preferred embodiment of the present invention there is provided apparatus for reading handwriting including wireless hand-held apparatus for sensing handwriting and providing an output indication of the contents thereof.

Additionally in accordance with a preferred embodiment of the present invention there is provided apparatus for reading handwriting including personally trainable hand-held apparatus for sensing motion during handwriting and providing an output indication of handwriting content.

Further in accordance with a preferred embodiment of the present invention there is provided apparatus for reading handwriting in real time comprising a hand held housing, a motion sensor disposed in the housing, recognizing apparatus disposed within the hosing and receiving signals from said motion sensor for sensing a plurality of handwriting characteristics and symbol recognizing apparatus disposed in said housing receiving the outputs of the plurality of parallel recognizers for providing an indication of a handwritten symbol.

Additionally in accordance with a preferred embodiment of the present invention there is provided apparatus for reading handwriting including hand-held apparatus for sensing motion during handwriting and providing an output indication of handwriting content in a form corresponding to that of a conventional keyboard.

Further in accordance with a preferred embodiment of the present invention there is provided apparatus for reading handwriting including hand-held apparatus for sensing motion during handwriting and providing an output indication of handwriting content in a RS-232 compatible form.

Additionally in accordance with a preferred embodiment of the present invention there is provided audio-visual apparatus including apparatus for providing a human sensible output including information in at least one of audio and visual form and having as an input element hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such audio-visual apparatus

include a video recorder and player, a stereo audio player and a television.

Further in accordance with a preferred embodiment of the present invention there is provided portable information storage and retrieval apparatus including a portable computer memory and output device and having as an input element hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such portable information storage and retrieval apparatus include a digital watch with memory, a computerized diary, a computerized dictionary and electronic telephone book.

Additionally in accordance with a preferred embodiment of the present invention there is provided lock apparatus including locking apparatus responsive to a predetermined electronic input and having as an input element hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such locking apparatus include door locks and vehicle door locks and ignitions.

Further in accordance with a preferred embodiment of the present invention there is provided magnetic card activated apparatus including apparatus for reading a magnetic card and having as a verification input element, hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such magnetic card activated apparatus include automatic teller apparatus and point of sales credit card acceptance units.

Various combinations of the above-mentioned structural and functional elements alone or in combination with additional elements, such as, for example, graphical input capabilities, are also within the scope of the present invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood and appreciated more fully from the following detailed description, taken in conjunction with the drawings in which:

Fig. 1 is a pictorial illustration of a handwriting reading device constructed and operative in accordance with a preferred embodiment of the present invention in an operative environment;

Fig. 2 is a simplified illustration of a preferred mechanical structure of the handwriting reading device of the present invention;

Fig. 3 is a simplified block diagram illustration of the handwriting reading device of Figs. 2 and 3;

Fig. 4 is a partially schematic, partially block diagram illustration of part of the apparatus of Fig. 3;

Fig. 5 is a block diagram illustration of part of the apparatus of Fig. 4; and

Figs. 6A and 6B are simplified flow charts illustrating operation of the handwriting reading device of Figs. 3 - 5 during handwriting reading. Fig. 6B illustrates the teaching process and Fig. 6B illustrates the recognition process.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

It has been found by the present inventor that each writer produces conventional alphanumeric characters from pen strokes selected from a set containing approximately 12 - 14 pen strokes, which set is characteristic of each individual. The present invention employs this realization to provide a personalizable handwriting input device. Appendix A contains a detailed exposition of the finding of the inventor.

Reference is now made to Fig. 1, which illustrates a handwriting input device 10 constructed and operative in accordance with a preferred embodiment of the invention in a typical operating environment wherein it communicates by wireless communication with a computer 11, such as a notebook PC having an associated receiver 12, such as a model RB 1023 RF receiver, commercially available from RF Monolithics, Inc. of Dallas, Texas. Receiver 12 may communicate with computer 11 via the RS 232 port thereof or alternatively via the keyboard input thereof.

The handwriting input device 10, which is illustrated in greater detail in Fig. 2, may be used on any writing surface or alternatively in the absence of a writing surface and does not require any special pad or writing substrate. Preferably the handwriting input device comprises a housing 13 in the general size and shape of an ordinary pen which is preferably provided with suitable indentations 14 for predetermined finger engagement.

Disposed in housing 13 is an ink reservoir and output point assembly 16, which may be constructed and operative in any conventional manner. Alternatively no ink output may be provided. In accordance with a preferred embodiment of the present invention there is disposed in a forward location of the housing 13 an accelerometer 20, preferably operative in three dimensions. Preferably the accelerometer 20 is located interiorly of indentations 14. A typical accelerometer which meets the size and power requirements of the invention comprises at least two and preferably three mutually orthogonally mounted Model 3031 accelerometers commercially available from EuroSensor of 20 - 24 Kirby Street, London, England.

Referring additionally to Fig. 3, it is seen that the output of the accelerometer 20 is supplied via an operational amplifier 24, such as a model LT1179, commercially available from Linear Technology Corporation of Milpitas, California, to a microcontroller 26, such as an Hitachi H8/536 microcontroller including an A/D converter. Microcontroller 26 is operative to extract a plurality of predetermined features of the acceleration sensed by accelerometer 20. It is a particular feature of the present invention that a relatively small number of discrete features derived from sensed acceleration during handwriting has been found to be sufficient to map the alphanumeric symbols of a given individual. It is appreciated that the characteristics of such features vary from individual to individual and accordingly, the microcontroller must be personalized through training in order to select suitable feature characteristics for a given individual.

A typical catalog of features extracted by the microcontroller 26 appears in the Tappert et al reference described hereinabove and is hereby incorporated by reference and also appears more explicitly in Appendix B. A preferred listing of software that provides the functionality of the microcontroller 26 appears in Appendix C.

The microcontroller 26 provides the functionality of a bank of parallel recognizers 26. The parallel recognizers are operative to recognize

128 different symbols. They may also be operative to recognizers various graphic symbols. The parallel recognizers are also personalized by suitable training which is preferably carried out using the pen 10 and the computer 11 and involves subsequent downloading to the pen 12, so as to associate the various alphanumeric symbols with given acceleration derived features extracted by the microcontroller 26.

The microcontroller 26 also provides the functionality of post-processing circuitry and is operative to select the most probable symbol from among those recognized by the bank of parallel recognizer and to encode it in a conventional universal code, such as ASCII, which is not in any way dependent on the personal handwriting characteristics of a given individual and which can be readily accepted by conventional computers.

Preferably, the coded symbol output from microcontroller 26 is in a form compatible with or identical to the output conventionally received at the keyboard input of a conventional computer, such as a PC.

In accordance with a preferred embodiment of the present invention, the coded output of microcontroller 26 is transmitted to computer 11 in a wireless manner by a wireless transmitter 32, such as a model MB1003, which is also commercially available from RF Monolithics, Inc. and which communicates with receiver 12 (Fig 1). Alternatively any other suitable IR transmitter or radio transmitter may be utilized. In such a case, the computer 11 is preferably supplied with a corresponding receiver which outputs directly to the keyboard input of the computer or any other suitable input, such as an RS 232 port. Alternatively a non-wireless communication connection may be provided. A suitable battery 33 is provided to power the apparatus located within housing 13.

It will be appreciated that the handwriting reading apparatus of the present invention is preferably a hand-held, personalized "pen" which can be carried by the individual for whom it has been personalized and used with any computer having suitable communication facilities. The computer need not be personalized in any way, inasmuch as all of the handwriting recognition hardware and software is resident in the "pen".

Reference is now made to Fig. 4 and Fig. 5 which illustrate portions of the apparatus of Figs. 1 - 3 in differing levels of detail. The accelerometer 20, as noted above, preferably comprises three separate accelerometer modules, each including a bridge containing piezoresistive elements outputs via operational amplifier 24 to microcontroller 26 including A/D converter channels 44. As noted above, the microcontroller 26 provides DSP functionality, represented by block 45 which is preferably operative to extract 32 features which are combinations of components of X-acceleration, Y-acceleration and up and down movements of the pen.

The output of functional block 45 is supplied to the bank of parallel recognizers 26 which includes a plurality of fuzzy comparators 46, each of which receives from reference feature storage facility 47 reference features for a plurality of alphanumeric symbols in addition to the serial input from DSP circuitry 24 which contains vectors representing the extracted features. The outputs of the fuzzy comparators 46 are supplied to corresponding error accumulators 48. The outputs of the error accumulators 48 are supplied to a fuzzy associative memory 50 which receives threshold inputs from a threshold definer 52. Threshold definer 52 receives data inputs together with control and timing inputs.

It is appreciated that in accordance with a preferred embodiment of the invention, the structure of Figs. 4 and 5 is embodied in software resident in microcontroller 26, as exemplified in Appendix C.

Figs. 6A and 6B are simplified flow charts illustrating operation of the DSP block 45 (Fig 4). Fig. 6A illustrates the teaching process and Fig. B illustrates the recognition process. The DSP block operates to perform double integration of X and Y acceleration to obtain not only velocities, but also positions and travel distances. In addition, non-linear positive and negative rotations are counted, located and their length recorded.

Appendix D sets forth an alogorithm for personalization of the handwriting reading device of Figs. 3-5.

The personalized functions are carried out using the pen 10 and the computer 11. The user initially writes each alpha-numeric symbol. The symbol is "read" by the device which determines X, Y and Z accelerations, and employs the accelerations to extract the features listed above, in a manner set forth generally in Fig. 6. Feature recognition is achieved by recognizing the centroids of the pen-strokes of the individual writing and by classifying of each character into 12-16 pen-stroke types. The character is considered to be a sequence of certain pen-strokes and is recognized as a fuzzy string over the pen-strokes alphabet. The fuzzy rules are downloaded to the microcontroller 26 in the pen 10.

The extracted features are employed to adapt fuzzy rules. The fuzzy rules are employed in the operation of a symbol recognizer which provides output indication of a recognized symbol.

The actual and recognized symbols are compared to provide a difference indication which is used to further adapt the fuzzy rules until an acceptable match between the actual and recognized symbol is attained.

Additionally in accordance with a preferred embodiment of the present invention there is provided audio-visual apparatus including apparatus for providing a human sensible output including information in at least one of audio and visual form and having as an input element hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such audio-visual apparatus include a video recorder and player, a stereo audio player and a television. Other home appliances such as washing machines and cooking apparatus may be operated along similar principles.

Further in accordance with a preferred embodiment of the present invention there is provided portable information storage and retrieval apparatus including a portable computer memory and output device and having as an input element hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such portable information storage and retrieval apparatus include a digital watch with memory, a computerized diary, a computerized dictionary and electronic telephone book.

Additionally in accordance with a preferred embodiment of the present invention there is provided lock apparatus including locking apparatus responsive to a predetermined electronic input and having as an input element hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such locking apparatus include door locks and vehicle door locks and ignitions.

Further in accordance with a preferred embodiment of the present invention there is provided magnetic card activated apparatus including apparatus for reading a magnetic card and having as a verification input element, hand-held apparatus for sensing motion during handwriting of the type described hereinabove.

Examples of such magnetic card activated apparatus include automatic teller apparatus and point of sales credit card acceptance units.

It will be appreciated by persons skilled in the art that the present invention is not limited by what has been particularly shown and described hereinabove. Rather the scope of the present invention is defined only by the claims which follow:

APPENDIX A

What the human brain tells the human hand
A behavioral perspective

Ehud Bar-On
E.B. Research & Development Ltd.
Guttwirth Bldgs, Technion, Haifa 32000, Israel

September 22, 1992

Abstract

This study investigates the primitives of motoric patterns of hand movement during handwriting. This is referred to as the "language" between the "hand and the "brain", and as such, has its own vocabulary and syntax. The "vocabulary" is the pen strokes and the syntax is how they are combined to pen strokes sequences. The handwriting is viewed as a high level cognitive activity of communicating, expressed as a complex motor skill and its investigation provides insight into the processes of chunking and automaticity. The main finding of this study is that pen strokes are specific to an individual writer, and characterize the writer's unique motoric control mechanism. The dynamic data of many thousand handwritten characters, produced by many writers, had been analyzed. The time domain signals were segmented into discrete pen strokes units and represented as vectors in a feature space. Those vectors were clustered, using a variety of clustering techniques. We found that in spite of the fact that the hand movements during writing could take any form or shape, a particular writer employs only a very limited set of pen strokes. The results of the clustering by various methods, yields a limited set of only twelve to fourteen types of pen strokes that accounts for 90brain supposedly chunks information to minimize the required attentional resources.

Keywords: handwriting, human motor control.chunking, automaticity, connectionism

1 Humans motor behavior

When we speak about "what the human brain tells the human hand", we speak about the kind of "motor control language", that might exist between the "brain" and the "hand". While this "brain-hand" communication can be approached from different points of view, we study it from a behavioral perspective. That is. we investigate the evidence of chunks or what is referred to as "motor programs", by analyzing the dynamic data collected during experiments in hand-writing recognition. Although all the results reported in this article consists only on the output of the handwriting process. we try to make the proposed model cognitive and biological plausible.

We'll start with the cognitive plausibility. There is a strong link between cognitive mechanisms and the human motor behavior. Handwriting is the way that humans express their thoughts through the use of a complex motor skill. Rosenbaum [12], presented handwriting as culmination of several internal translation process. First, an abstract message or idea is constructed. Then, it is formulated into appropriate linguistic expression, and then translated as a series of efferent commands. There is a basic similarity between speech and writing, so we can assume that both share the same underlying mechanism. The phonemes in speech, correspond to pen-strokes in writing, and the morpheme in speech correspond to letters. The higher levels of abstraction (i.e. words, syntax, lexicon, semantics, prosody and discourse), are probably the same.

There is an empirical evidence, that the ability to sequence behavior, whether in the linguistic domain or drawing domain, depends on a central, amodal mechanism [11]. If this is true, then motoric control should be considered as obeying the same rules of the linguistic or other sequential cognitive mechanisms. On the other hand, some of the properties that we discover about the primitives of the motor control language, might be generalized to other cognitive activities. In the case of handwriting, the efferent commands are expressed as pen-strokes. It was found by Wright [17], that different production mechanism are probably controlled by the same high-level graphic representation. This view of hierarchical structuring. and a "virtual" representation of motor movement is supported by empirical evidence. Humans can write in a consistent style when they write in small letters in their notebooks, or when writing much bigger characters on a blackboard. Moreover, people can write with a consistent style (same pen-strokes), when using different effectors like hand and foot.

As in any behavioral research. resulting behavior is influenced both by the general properties of handwriting and by properties which are specific to an individual writer. This means that a considerable part of the variance can be attributed to individual differences. It has been noticed by many researchers that handwriting style is so distinctive. that writers can be recognized according to their hand writing. This is also a common knowledge, and therefore signatures are recognized as a unique identifier of a specific writer. Some theories even associate personality traits with hand-writing style. As we intend to show in this article, the primitive pattern of writing, are unique for individuals writers.

As any other cognitive or motor activity, human motor control goes through a process of development that is equivalent to intellectual development. An interesting well known phenomena is that children's drawing (and after that writing). becomes more refined over the course of development. It has been suggested by many researchers, that early drawing behavior correlates with young children's cognitive abilities. Van sommers [15], suggested that drawing may be governed by high-level rules, similar to those governing language processing, and that the development in drawing may parallel the development of language. Goodnow and Levine [9] even suggested a: "Grammar for action: sequence and syntax of children's copying." They reported several rules for sequencing drawing strokes. Examples of such rules were:" Start at leftmost point", "Start at top", "Start with vertical strokes", "draw horizontal lines from left to right", etc. The evolutionary rationale for such rules could be to simplify motor planning.

The biological plausibility of an hand writing model, involves two parts: The plausibility of the assumed neurological control, and the biomechanical properties of the hand. The preservation of the writing style while using different muscles and even organs, is one of the most intriguing questions. The automaticity of writing, suggests a chunking mechanism, but this chunking mechanism is probably not in the motoric system of the hand, but somewhere in the upper control levels of the brain. Therefore. whenever we refer to the "hand", we do it metaphorically. I.e. the "hand" represents the efferent mechanism that accomplishes the motoric control. Recently, Alexander et al. [3], raised the question whether the specific concept of a "motor program", is an appropriate foundation for the development of biological plausible models of how the brain controls movements. While our current knowledge about the cortical and basal ganglia motor areas is still far from allowing a specific model, it can suggest what models are more neurologically plausible than others. Fischbach [7] discusses the finding about "face cells" and "motor command cells" as an evidence for abstraction in the brain. In the monkey's visual system, "face cells" located in the inferior temporal sulcus, were suggested as representing a high level of abstraction. These neurons respond to faces but not to other visual stimuli. Face cells have their counterparts on the motor side. "Command" neurons have been identified in certain vertebrates that trigger fixed action patterns. Georgopolos [1] recorded electrical activity of single neurons, and found command neurons in the monkey's motor cortex (precentral gyrus) that encode the direction of forelimb movement. The firing of these neurons was not associated with the contraction of a particular muscle or with the force of the coordinate movement. Georgopoulos computed a vector by summing the firing frequencies of many neurons, and found that it is more correlated with the direction of movement than is the activity of any individual cell. The vector becomes evident several milliseconds before the arm moves. He interpreted this result as evidence for motor neuron planing. Damasio and Damasio [4] discussed the linguistic behavior of patients with lesion in the left posterior temporal and inferior parietal cortex. It was found that such patients have problem in producing word forms from the available phonemes. Analyzing the accumulated empirical finding on language structures, gathered with assistance of imaging techniques like MRI (Magnetic Resonance Imaging) and PET (Positron Emmision Tomography), shows that linguistic activity like naming, involves the motor cortex activation together with anterior and posterior language centers in the left hemisphere. Writing is a language activity which involves a production center that forms words and activates the "command cells" in the motor cortex to produce pen-strokes sequences (letters) and written words. In the same way that speech is composed of a small set of phonemes, we argue that handwritten letters are composed of a small set of pen-strokes.

In addition to the neurological plausibility, there are biomechanic constraints on the "hand" part. Some general principles have been suggested as governing this control mechanism. For example, Flash and Hogan [8], proposed that humans tend to write in a way that minimizes *jerk*. That is, the third time derivative of the position signal. A more recent study [6], suggested the *snap*, which is the fourth derivative of position, as the cost function that is minimized. As we will see, there are alternative hypotheses about the type of constraints imposed on the biomechanics of handwriting. In spite of the fact that the principle that governs the handwriting might be universal, each writer has its own unique variation. The differences are more pronounced in the unwritten strokes (the pen movements that do not touch the writing surface0 than in the written ones. The friction of the pen with the writing surface diminishes the characteristics of the hand control mechanism which are better revealed when the pen is up.

The article starts with reviewing theories of "motor programs", and arguing against that term and what it implies. We'll propose an alternative connectionist model of primitive hand-writing patterns and argue that it is more biologically and cognitive plausible. Then, we describe the experiment, and the collection of the data. The fourth chapter will describe the analysis of the collected data, and the conclusions that were drawn from this analysis. The last chapter will discuss the results, and compare our conclusions to alternative ones. We shall conclude the article by pointing out some future directions and implications of the suggested model.

2 Attention, chunking and "motor programs"

The concept of "working memory" is modeled after the "working memory" in a (von Neumann) computer, where the registers in the Central Processing Unit (CPU) have a similar function. This is also why researchers in that field prefer to talk about "motor programs". "Motor programs" are supposed to save attentional resources. According to this approach, it is assumed that the brain controls movements like handwriting. by executing "motor-programs", much like software is used in a computer (e.g. [12]). The "motor program" concept is attractive, as it reduces the complexity of the sequential, analytical approach by using pre-programmed sequences of a limited number of generic motor commands (or routines), to control a large repertoire of movements. Alexander et al. [3] points out difficulties with the neurological plausibility of "motor programs" that imply separation between "software" and "hardware". For example, what would constitute the software in such a model, and where it is stored when not executed, how are they assembled prior to their execution and how new programs are created. A major problem with the "motor program" approach is also the sequencing of performance: goal- directed movements are supposed to be translated into trajectories, then to joints kinematics. Muscle activation cannot be computed until the inverse dynamics is calculated and so on. Therefore, argue Alexander et al., signs of specialization for such transformations should have been found in the cortical and basal ganglia. Thusfar, neurobiological evidence seems to indicate lack of such specialization.

It is assumed that the processing capacity is limited, and therefore several tasks that have to be carried out simultaneously compete on the same resources. The main "problem" of human-beings and other organism might be, that we lack a "parallel output channel". All the output channels, be it speech, handwriting or any other motor output, are all serial in nature. It might be that this serial output suggested a serial cognitive mechanism as well. It is the conjecture of the neural-nets literature, and of this article, that the underlying mechanism is parallel and distributed over millions of simple processing units (neurons). Therefore, the term "motor programs", that implies a serial symbolic process running in the "brain-computer" might be misleading. We prefer to speak about "motoric schemas", which are motoric patterns invoked by activation of an assembly of neurons.

The connectionist view of schemas (Smolensky, 1986) is that stored knowledge-atoms are dynamically assembled at the time of inference, into context-sensitive schemata. Rumelhart and McClelland (1986) [14] proposed a technique that suggests how an attentional selective mechanism might work.

They propose the use of a set of mapping units which produce "dynamically programmable connections" and achieve focusing on different features on different times. Smolensky (1986) maintains that schemata are coherent assemblies of knowledge atoms, where coherence or consistency is formalized under the name of harmony. He proposes the harmony principle: the cognitive system is activating coherent assemblies of atoms, and draws inferences that are consistent with the knowledge represented by the activated atoms.

In much the same way, we propose to speak about "motoric schemas". This is consistent with our conjecture that there is no essential difference between the so-called "cognitive" and "motoric" brain mechanism. The connectionist schema-model is also consistent with the neural evidence, that the specialization among different cortical motor areas are related to certain sequences of movements, and not to transformations as proposed by the "motor program" literature. According to our conjecture, preparatory units and movement executing units will belong to the same schema. This is supported by the anatomical fact that the three motor areas (SMA - Supplementary Motor Area, PMC - Primary Motor Cortex and Putamen), has the same proportion of target dependent motor cells and limb dependent movement-related cells ([2]). Another supporting evidence is that neuronal population that were supposed to represent different stages of computation (according to the "motor program" view), have been shown to be active simultaneously.

3 The experiment

3.1 Data collection

Rumelhart [13] developed a system which learns to recognize cursive script as it is generated by a writer. This system learns from examples of cursive script produced by a number of writers and recorded. He collected approximately 1000 words from each of 58 writers. The average length of a word is about 8 characters, That sums up to nearly 500,000 examples of handwritten cursive characters. His results were encouraging and had been used in this research. While Rumelhart [13] was mainly interested in handwriting recognition, this article uses the same data to investigate the writing mechanism.

The data were collected in the following manner. Each word in the corpus was recorded. It was then played to the writer who was instructed to write the word on a tablet digitizer. The resulting x coordinate, y coordinate and an indication of whether the pen was or was not on the paper were sampled each 10 milliseconds. The resolution (more than 200 dpi) and the sampling rate (100 samples/sec) are those that are shown to be appropriate in the on-line hand-writing recognition literature ([16]). The data was saved as files, and has been used for the analysis reported in this article.

In addition to the data from Rumelhart's experiment, several thousands pen strokes of Japanese handwriting were collected. Most of the data has been collected from hand written Hiragana characters, but some data has been collected during writing Kanji (idiographic) Japanese characters. Hiragana characters has the curved shapes of english hand printed characters, but without the ligature of cursive handwriting.

Preprocessing of the hand-writing raw data has been made. with the goal of extracting features that will be used to segment and characterize the "pen-strokes". A pen stroke was defined as a segments of the cursive writing signal, between two consecutive zero crossing of the vertical velocity of the pen movement. Each character was segmented to several segments or "pen-strokes". A typical writing rate in English is two letters per second. Writing Japanese characters (Hiragana), takes about the same time, and a typical Hiragana character can be written in 0.3 - 0.5 seconds.

3.2 Segmentation and feature extraction

The principle of segmentation and feature extraction is to segment the continuous signals into discrete segments and to represent each segment by a feature vector in the feature space.

The segmentation that produces "pen-strokes" out of the continuous signals, depends on the different definitions of the term"pen-strokes". While most of the literature about on-line character recognition is using this term, there isn't an agreed upon definition of a "pen-stroke". For example, one often finds only the pen-state change as the only criteria. That is. definition of a stroke as continuous pen-movement. between pen-down and pen-up consecutive states [16].

Once a "pen-stroke" is defined. there are many ways to represent it in a feature space. The on-line character recognition research employs several orthogonal transformations such as a discrete Fourier transform of the curve segments corresponding to the pen-strokes. That is, a pen-stroke is represented by its Fourier coefficients obtained from its $x(t)$ and $y(t)$ signals. Essentially, any orthogonal transformation (e.g. Walsh transform, Karhunen-Loeve) could do in approximating the pen-strokes curves. That is, Plane curves can be approximated by orthogonal functions (Sinusoidal, polynomial or even square waves). This description can be also easily converted to the frequency domain. as was done in several studies of hand-writing recognition [16].

This mapping of the time domain to a parametric domain is advantageous when the characters can be represented by a small number of coefficients. Therefore, periodical smooth curves lend themselves better to modeling by harmonic functions, as one needs less coefficients. On the other hand. straight line strokes require high order harmonies as they include high frequency components. This is why sinusoidal approximation is useful for characters consisting of curved strokes, as found in English cursive script, more than for Japanese Kanji characters (that are made mainly of straight line segments). K-L expansion has been proved to be a successful algorithm in machine-printed Chinese character recognition. Another successful attempt was to use a modified Hough transform for recognition of Chinese hand-written characters. The Hough transform is a technique for line detection and has been generalized to detect arbitrary shapes. Chinese characters are line-like, and therefor lend themselves naturally to a Hough transform representation.

The segmentation and feature extraction methods depend of course, on the goal. If the goal is pattern recognition, then the segmentation and feature extraction are geared toward discrimination between the various patterns. In our case, we looked for a segmentation and features that are biological plausible. Consequently, we investigated only features that might be explained by the neurobiological control structures, like the direction of the strokes, their curvature etc.

3.3 Hollerbach's model

The segmentation and feature extraction mechanism employed was, to develop a model of the underlying handwriting process and to describe the data in terms of the parameters of the model. The model employed was derived from that of Hollerbach [10] and involved the assumption that the generation process could be described as pair of coupled oscillators. The coupled harmonic oscillators is just one of the many models that exist. Actually, its basic assumption about the symmetric shape of the velocity profile (an half sinus shape), is probably an oversimplification. The literature about velocity profiles of pen-strokes usually assumes an asymetrical bell-shaped velocity profile. That is, a rapid-aimed movement described by a log-normal velocity profile is considered as the fundamental unit (stroke). More complex movements are described in terms of superimposed log-normal curves. The asymmetric nature of the velocity bell-shaped profile results from the global stochastic behavior of a large number of processes involved in velocity control.

In spite of being an oversimplified and inaccurate model, it has a clear advantage that it is based on a control mechanism, and is neuorobiologically interpretable. This model assumed that:

$$\dot{x} = a\cos(\omega_x t + \phi) + c \tag{1}$$

$$\dot{y} = b\cos(\omega_y t) \tag{2}$$

In words, the idea is simply that writing involves two orthogonal pendular movements. If we speak about writing in a notebook (small size letters), we can think about the wrist horizontal movements (actually, it is more arc-like movements) and the fingers flexion andextension vertical movement. These two movements can be considered as independent. If the size of the letters is more than an intch, than the arm muscles are involved.

According to this model, The y-axis consists of a series of up/down strokes whose velocity profile is assumed to be sinusoidal. The x-axis is also pendular with a constant velocity, c, to the right. Different characters are made by modulating the relative amplitudes, a and b, the relative phase, phi, and the relative frequency ω_x and ω_y , in the x and y directions. It is, furthermore assumed that the parameters change only when the velocity in the y direction reaches zero (end of pen-stroke). Thus, we define a *stroke* as the motion between zero crossings in the y velocity - v_y. In addition, segmentation occurs when the pen-state changes (from pen-down to pen-up or vice versa).

It should be stressed that Hollerbach's model was designed for synthesizing handwritten-like character, by a second order mechanical system. This model does not try to imitate the human motor control, or to be used for analysis of human handwriting. However, as it is a control system model, some of the parameters might be interpreted in terms of the human biomechanical system. For example, the parameter ϕ, which designates the phase shift, can be interpreted as relating to the delay in the nervous-muscular control system. As such, it can have an important diagnostic value in motor diseases.

When it was applied by Rumelhart to handwriting analysis, it suffered from some drawbacks. One of them is that the model is fitted not to the image, but to the velocity profile of the stroke. This simplifications tend to work well in most of the cases of English cursive hand-writing, because of the periodical nature of the v_y velocity signal.

As the Hollerbach model that we used, is based mainly on the velocity signals, we will illustrate the transformation from the $x-y$ domain of the hand-

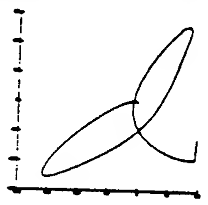


Figure 1: The handwritten letter d

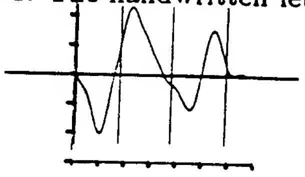


Figure 2: The v_x graph for the handwritten letter d

written character, to the corresponding velocities. Examples of the handwritten letter d, the corresponding velocity profiles and the reconstructed d are shown in Figures 1,2,3 and 4.

As can be seen, the reconstruction isn't perfect, and the curvature of the first pen-stroke of the d is opposite to the original. This result illustrates the fact that the model tries to reconstruct the velocities and not the resulting pen strokes image.

This does not exclude the fact that sinusoidal approximation worked for Rumelhart in recognition of cursive script. It turned out that in some cases (periodic signals during cursive handwriting in English) the model worked satisfactorily.

The Kanji characters, on the other hand, have more short straight segments, as can be seen in the following figures:

The "mori" Kanji character in the picture, is segmented to 27 pen-strokes (the last two pen-down strokes in the third "tree" are missing). Sixteen out of the twenty seven, are strokes in which the pen touched the paper, and 11 were just for moving the pen from one line to the other. Twelve sequences of "pen-down" strokes, correspond to the visible line segments in the character.

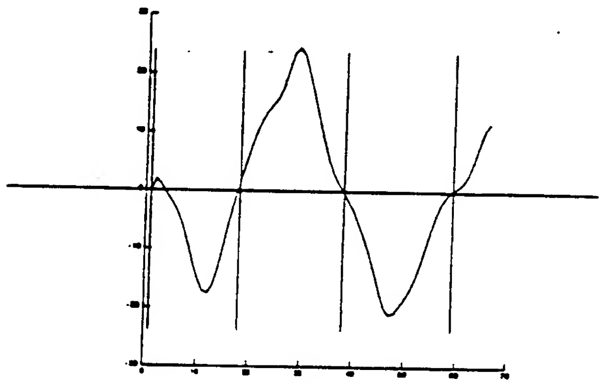


Figure 3: The v_y graph for the handwritten letter d

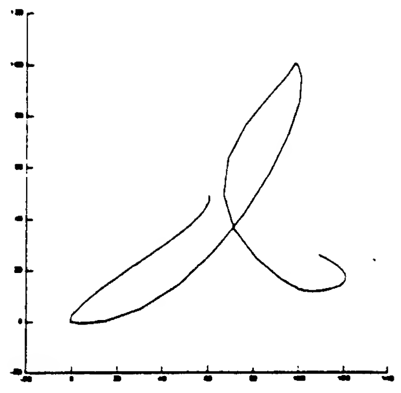


Figure 4: The reconstructed letter d

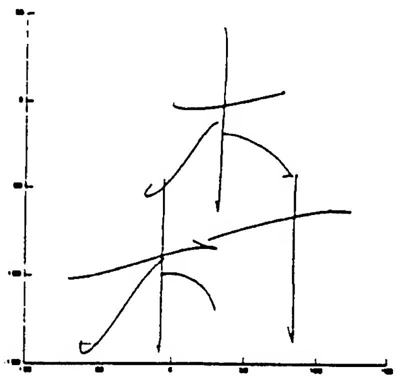


Figure 5: The separate strokes are more evident in Japanese Kanji characters. This is the Kanji character: "mori", which means: forest

y	x	Mid	Velocity	Harmony	Pen up-down
0.70	7.41	0.74	-2.95	4	1
3.44	-3.86	-2.77	-2.20	4	0
-10.56	-0.45	-0.01		2	1
0.71	-0.13	0.04	-0.33	2	1
4.40	0.08	-0.59	2.62	2	0
-4.20	-4.94	-1.88	-3.19	2	1
3.87	4.61	1.55	5.46	2	0
-2.67	5.51	3.63	-2.33	2	1
-5.74	-15.08	-7.31	-4.36	2	0
1.64	9.70	2.62	8.60	2	1
0.42	-1.46	0.23	-4.40	2	1
3.27	-1.99	-1.85	0.75	2	0
-9.71	-0.19	-0.04	-0.20	2	1
5.45	0.47	-1.20	3.70	4	0
-5.40	-5.85	-3.39	-2.74	4	1
4.29	4.89	0.84	10.50	2	0
-1.91	4.29	3.38	-1.56	2	1
3.91	-0.64	1.55	-3.30	4	0
1.82	9.72	3.70	6.74	2	1
0.07	-1.35	0.46	-5.25	2	1
2.00	-2.17	-2.12	1.86	2	0
-9.60	-0.18	-0.18	-0.28	4	1
0.98	-0.37	-0.09	-0.74	2	1
4.52	0.57	-0.33	-0.12	4	0
-4.22	-5.45	-2.72	-2.64	4	1
4.14	6.24	4.40	5.90	4	0
-5.10	7.23	4.61	0.54	4	1

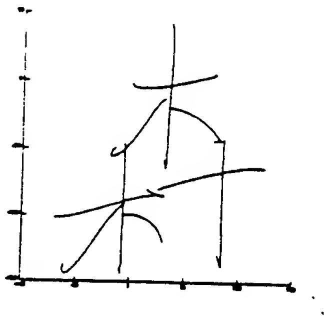


Figure 6: v_y signal for the Kanji character: "mori".

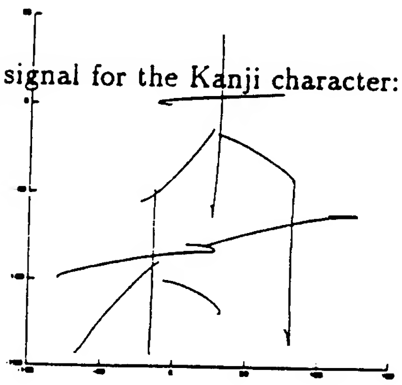


Figure 7: The reconstructed Kanji character: "mori".

The reconstructed Kanji character is depicted in the figure.

3.4 Recognizing pen-strokes sequences

One of the key problems in recognizing cursive handwriting is the segmentation problem. Rumelhart [13] has devised a learning algorithm for cursive handwriting recognition which combines word recognition and letter recognition. The letter recognition is based on recognizing PMPs, and PMPs sequences make letters. This system involves simultaneously learning to recognize and segment letters.

Although Rumelhart's experiment was done for hand-writing recognition, there are several things that can be learned from it, concerning the PMPs and their sequencing during handwriting. It was recognized in the early sixties

([6]), that motor knowledge can be used in recognition of hand-writing. A system called *Analysis by synthesis* suggests that characters are recognized by their "motor programs". These "motor programs" are supposedly deduced by guessing an initial program, and iteratively updating it according to the difference between the synthesized and actual forms. The connection between reading and writing process have been corroborated by the co-occurrence of certain kinds of acquired dysgraphia and dyslexia ([5]). In contrast to earlier approaches, Edelman et al., ([6]), assumes that while readers use motor knowledge in reading, they do not seem to do so by mentally reproducing the process of writing. The connectionist model that we propose isn't bothered, of course, by those distinctions between explicit simulation or implicit knowledge. This is another example of the misleading influence created by the "motor programs" metaphor.

4 The results of clustering

4.1 fixed radius clustering

The basic units of clustering were the pen-strokes. each of which was represented as a point in an n dimensional space. Out of the six features that we extracted for each stroke only three have been used. First, we used only one frequency for the modeling, so the rare strokes that involved higher harmonies were removed. Second, we did not differentiated between Up-strokes and Down-strokes. Up strokes contain more high order harmonies, but we limited our analysis to the basic movements, and tried to ignore the fluctuation induced by the bio-mechanical control mechanism. The third feature that wasn't used was the mid-point. For the reconstruction of the pen-strokes in the spatial domain, the x-coordinate of the midpoint in each stroke was computed. However, our preliminary analysis showed that this variable was very highly correlated with the x variable. This preliminary analysis, yielded three variables that were almost uncorrelated: Δ_x, Δ_Y and velocity. The dimension of the space were:

1. Δ_y - The relative displacement on the vertical direction.
2. Δ_x - The relative displacement on the horizontal direction.

3. v - v The v_x velocity at the end of the stroke is calculated according to equation below.

For a pen-stroke between a and b, which is approximated by a certain oscillation frequency, we calculate:

$$I_x = \int_a^b dx \cos(2\omega_x t) \quad (3)$$

$$v = [\frac{x_a - x_b}{l}] - [\frac{2I_x}{l}] \quad (4)$$

$$(5)$$

It is calculated in a different way for different ωt and this is an example of such a calculation.

The clustering of data from many writers, didn't yield satisfactorily clustering, but the clustering of individual writers did. E.g. the clustering for a particular writer, revealed 13 compact clusters that contained 90to 14, were consistent to all the writers we analyzed. This is a corroboration to our conjecture that hand writing is made out of a small number of PMPs, which are unique to an individual writer.

The centroids of the clusters, were reconstructed from the feature space, and are displayed on a 2-D spatial domain. As can be seen clearly from the results, different writers have different stroke types:

As each writer has about 25,000 pen-strokes that we wanted to cluster, we started with a fast clustering algorithm, similar to the k-means algorithm. The main requirement of the clustering algorithm were that it will be able to deal with very large data sets and find satisfactory clusters in few (2-3) iterations. The other requirement, which was even more important, was that the centroids will be good representations of the observations within each cluster. This requirement lead to seeking compact, hyperspherical clusters, that do not exceed a predefined radius. Elongate clusters are therefore represented by several adjacent clusters. Those clusters will be merged in a latter stage by an hierarchical clustering algorithm.

The clustering employed a two phase strategy. First, a fast "nearest centroid sorting" algorithm was employed to reveal the clusters in the large data set. Then, the resulting centroids of the clusters have been submitted to different hierarchical clustering methods. The first phase algorithm was

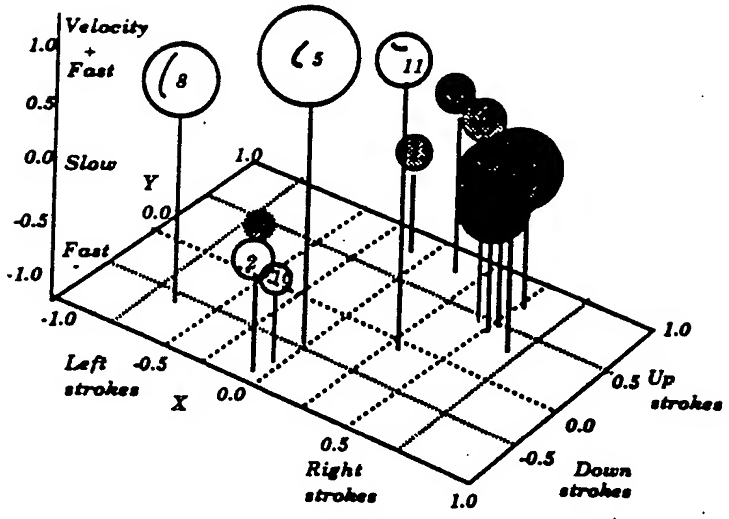


Figure 8: Clustering of 25,000 strokes of the same writer. Gray clusters represent down strokes.

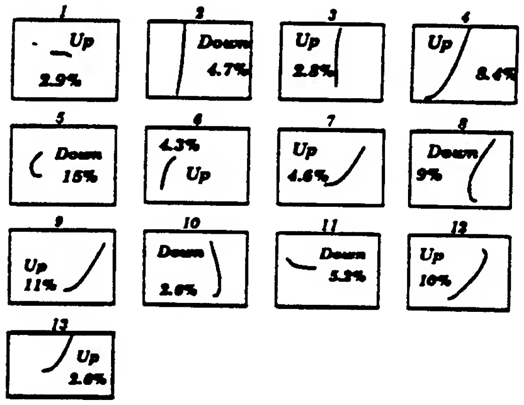


Figure 9: Thirteen centroid pen-strokes of an individual writer, including their relative frequencies.

sensitive to outlier strokes, that formed separate clusters. This was the reason why we got many very small clusters. These clusters accounted for less than 10of the observations. They were considered to be noise, or very exeptional pen strokes, and have been removed so not to influence the representativeness of the centroids of the large clusters.

The second phase included clustering of the resulting centroids using ten different methods. We distinguished between methods that yield compact hyperspherical clusters, and those that can detect elongate clusters. We start with the first group of eight clustering methods:

1. Average Linkage cluster analysis
2. Centroid hierarchical cluster analysis
3. complete linkage cluster analysis
4. Equal variance maximum likelihood method
5. Flexible data cluster analysis
6. McQuitty's similarity analysis
7. Median Hierarchical cluster analysis
8. Ward's minimum variance cluster analysis

The different methods tend to favor different characteristics such as size, shape or dispersion. For example, methods based on the least-squares criterion such as k-means or Ward's minimum variance method, tend to find clusters with roughly the same number of observations in each cluster. Average linkage is biased toward finding clusters of equal variance. Most clustering algorithms, except for single-linkage and density-linkage, tend to produce compact, roughly hyperspherical clusters. The clustering methods which are based on nonparametric density estimation, like the single linkage, will be discussed later in this chapter.

All the above clustering methods yielded very similar results, and the tree-based partition was essentially the same. The use of many different algorithms has been employed to investigate the robustness of the clustering structure under different hierarchical clustering methods. The result of all

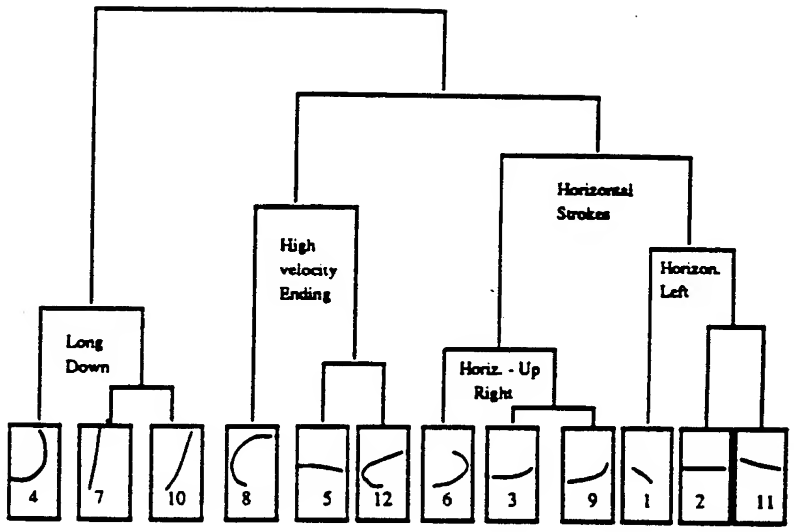


Figure 10: Hierarchical (compact) clustering of the 12 pen-strokes centroids of a particular writer

the above method revealed the following tree-based partition of the set of the basic twelve pen-strokes (of a particular writer).

From looking at the results of the hierarchical clustering, there is an obvious super-clusters that emerge. The horizontal-left strokes are one such a group, long down strokes are another group. In general we see a distinction between horizontal strokes and vertical strokes. The horizontal strokes themselves are subdivided to horizontal-left directed strokes, and horizontal right and up directed strokes. The high velocity C shaped strokes are part of circles or ovals. It should be noticed that for a specific writer, a certain stroke is always accomplished in the same way. For example, an horizontal short stroke, like crossing a t, will be done always as left directed strokes. Someone else could use only horizontal right directed strokes for that purpose. However, it is very unlikely that the same writer will use both an horizontal-left and horizontal-right strokes. The same is true with long vertical strokes. Once the writer is using a long vertical down-stroke, he will produce vertical lines always as down strokes of the same type and velocity profile. This organization of pen strokes was consistent in all the hierarchical

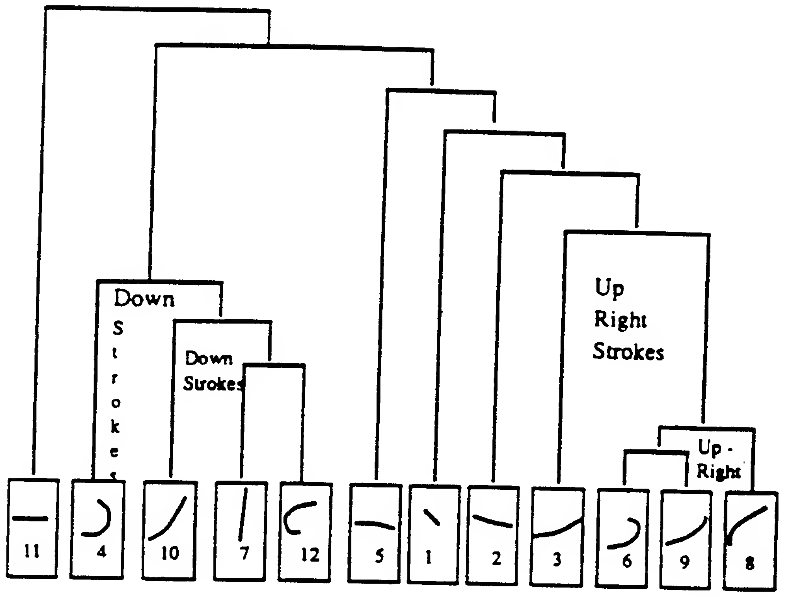


Figure 11: Hierarchical (Density linkage) clustering of the 12 pen-strokes of the same writer

clustering algorithms that we mentioned above.

The clustering methods that employ nonparametric density estimation, like the "Density linkage cluster analysis", can detect also elongated cluster shapes. These clustering techniques yielded two distinct super clusters: the "down and long pen-strokes" , and the "up and right strokes". The down strokes are those that form the "back-bone" of the English characters, while the up-right strokes are typically those that are used as ligature.

4.1.1 The "characteristic" shape of pen-strokes

As was argued above, any writer has a specific set of pen strokes that characterize the writer. While the same writer will have similar pen-strokes, in writing different languages, the frequency of appearance of a specific pen strokes depends, of course, on the language. In order to characterized a specific writer, in respect to her/his pen strokes, we suggest the "Pen-strokes Ordering Diagram" (POD). Such PODs are displayed in the following figures.

In spite of their strange looking, those diagrams are quite valuable, and convey important information about the handwriting of the analyzed writer.

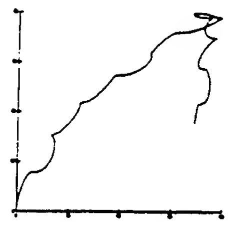


Figure 12: The centroids of the pen strokes of a writer, for English cursive writing. The pen-strokes are ordered according to their v_y values, from up-strokes to down strokes

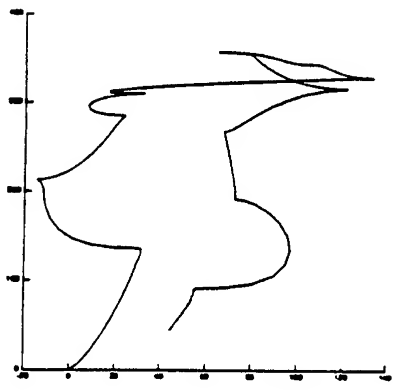


Figure 13: The centroids of the pen strokes of a Japanese writer, for Japanese Hiragana characters. The pen-strokes are ordered according to their v_y values, from up-strokes to down strokes

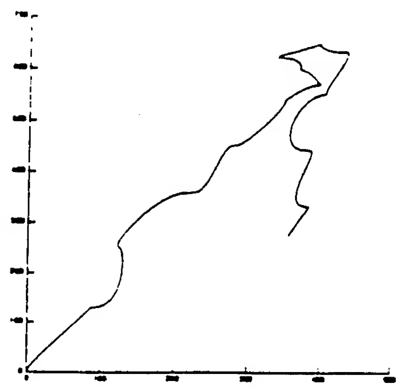


Figure 14: The centroids of the pen strokes of a Japanese writer, for english characters. The pen-strokes are ordered according to their v_y values, from up-strokes to down strokes

There is a very clear distinction between the key strokes that these two writers are using. This is true to other writers as well. Each writer uses a unique set of pen-strokes: different slopes, different curvetures, different velocity profiles and accelerations.

5 Discussion and future research

We will start our discussion with comparing the conclusions of Rumelhart's handwriting recognition experiment, and the conclusions of this study. In Rumelhart's handwriting recognition experiments, both writer dependent and writer independent recognizers have been trained. Two networks have been trained to recognize the writing of individual writers and one network has been trained on four different writers as a "writer independent" recognizer. On the writer dependent networks Rumelhart found that, for a vocabulary of 1000 words, on words never seen during training that 99top five, approximately 90On the writer independent data the results are somewhat worse. That is, about 70

According to the results in this study, we have a basis to doubt this conclusion. The inter-writer variability is too big, and more writers will not lead necessarily to better results.

Another finding of Rumelhart was that writers can be trained easily to produce recognizable hand writing. He developed an "online" system in which the network recognizes (and can be trained) as the writer writes on the digitizer. With a little care on the part of the writer it is not difficult to achieve a score of better than 90correctly classified on the writer independent system. (It is also possible to write so that the recognizer does much more poorly than that.) Careful experiments on a person's ability to adjust to the recognizer have not been carried out. By limiting the vocabulary to one hundred words or less, it seems to be possible to obtain near perfect performance. (It will, of course, depend on the confusability of the words.)

The main conclusion was, that it would be useful to embed the recognizer in a network of networks each trained on a subset of the writers - perhaps one for printers, one for pure cursive writers etc. This line of thought led to the current study, reported in this article. That is, that it may be useful to study the individual differences among the writers. The idea of studying individual differences, as a mean towards better handwriting recognition, turned out to start a new line of research - the study of writer's unique pen-strokes, which is related to the topic of automaticity in brain - hand communication.

This study started from that point. The main question that we posed was if individual writers have distinct sets of pen-strokes, which are consistent and well defined. The reanalysis of the data from this perspective encourage to believe that this is the case. Human writers have 12-14 distinct pen-strokes, which are characteristic for a certain writer. These pen-strokes are the primitive "motoric patterns", of which handwriting is composed. We showed also that the primitive pen-strokes cluster to super-clusters, thus revealing the hierarchical nature of the control mechamnism. These findings are consistent with the neurological literature, that we cited in the introduction. That is, there might be "command cells", that get the activation for certain words (letter combinations) from another center in the brain and activate pen-strokes mechanism. The pen-stroke is controlled by a direction and amplitude cells, that activate the corresponding primitive motoric patterns (PMPs). The next stage will be to locate the cell regions that are responsible for this activation in the motor areas of the human brain. This is under research now with the help of Magnetic Resonance Imaging (MRI) method, when the MRI is tuned to detect cerebral blood flow. We would expect that the learning to write, should show itself as forming of such motoric activation centers, corresponding to what we have found in this study.

Those findings have implications to the study of automaticity and chunking. One question to be investigated is if the motor control mechanism is central and a modal, as suggested by previous researchers. This can be investigated by studying the patterns of interference between modalities. For example, an experiment in which the subject is instructed to pronounce one character, and write another character at the same time. In addition to predicting longer Reaction Time, we can now predict interference between the pen stroke patterns and sequences. Another interesting question is how are the motoric patterns stored and how are they retrieved when needed. Our conjecture, which is consistent with the neural net model, is that the retrieval time will be independent of the number of patterns sequences. Some support for this conjecture is that it takes the same time to write a character in a large character set (kanji) or small character set language (Hiragana, English).

Future research that will combine behavioral analysis with neurobiological research, might answer many of the questions that we raised in the introduction.

References

[1] R. E. Kettner A. P. Georgopoulos and A. B. Schwartz. Primate motor cortex and free arm movement to visual targets in 3-d space. *Journal of Neuroscience*, pages 2928–2937, 1988.

[2] G. E. Alexander and M. D. Crutcher. Neural representations of the target of visually guided arm movements in three motor areas of the monkey. *Journal of Neurophysiology*, 64:164 - 178, 1990.

[3] G. E. Alexander, M. R. De-Long, and M. D. Crutcher. Do cortical and basal motor areas use motor programs to control movement? *Behavioral and Brain Sciences*, 1992.

[4] A. R. Damasio and H. Damasio. Brain and language. *Scientific American*, 267(3):88 - 95, 1992.

[5] S. Edelman. *Reading and writing cursive script: A computational study*. PhD thesis, Weizmann Inst. of Science, Rehovot, Israel, 1988.

[6] S. Edelman, T. Flash, and S. Ullman. Reading cursive handwriting by alignment of letter prototypes. *International journal of Computer Vision*, 5:303-305, 1990.

[7] G. D. Fischbach. Mind and brain. *Scientific American*, 267(3):48-57, 1992.

[8] T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *journal of Neuroscience*, 5:1688-1703, 1985.

[9] J. J. Goodnow and R. Levine. The grammar of action: Sequence and syntax in children's copying. *Cognitive Psychology*, pages 82-98, 1973.

[10] J. M. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:139-156, 1981.

[11] S. W. Keele. Sequencing and timing in skilledv perception and action: An overview. In D. Allport, W. MacKay, W. Prinz, and E. Sheerer, editors, *Language perception and production*. Academic Press, London, 1987.

[12] D. A. Rosenbaum. *Human motor behavior*. Academic Press, 1991.

[13] D. E. Rumelhart. Segmenting and recognizing online cursive handwriting. Technical report, PDP Lab., Stanford University, 1992.

[14] David E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, chapter 8, pages 318-362. MIT Press, 1986.

[15] P. Van Sommers. *Drawing and cognition: Descriptive and experimental studies of graphic production processes*. Cambridge Univ. Press, Cambridge, England, 1984.

[16] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line hand writting recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(8), 1990.

[17] C. E. Wright. Generalized motor programs: Reexamining claims of effector independence in writing. In M. Jeannerod, editor, *Attention and performance XIII*, pages 294–320. Lawrence Erlbaum, Hillsdale, N.J., 1990.

APPENDIX B

```
/************************************************************** ...
 feat[0]  = relative size of symbol 32*div_round(Xwidth,Ywidth)
 feat[1]  = number of points
 feat[2]  = number of strokes
 feat[3]  = number of spaces
 feat[4]  = number of different levels in X for end of space
 feat[5]  = number of different levels in Y for end of space

 feat[6]  = number of Ext X in neg.rot
 feat[7]  = different levels in X for minX
 feat[8]  = different levels in Y for minX

 feat[9]  = number of Ext in X pos.rot
 feat[10] = different levels in X for maxX
 feat[11] = different levels in Y for maxX

 feat[12] = number of Ext in Y neg.rot
 feat[13] = different levels in X for minY
 feat[14] = different levels in Y for minY

 feat[15] = number of Ext in Y pos.rot
 feat[16] = different levels in X for maxY
 feat[17] = different levels in Y for maxY

 feat[18] = number of ExtXY  for neg. rot
 feat[19] = different levels in X for feat 41
 feat[20] = different levels in Y for feat 41

 feat[21] = number of ExtXY  for pos. rot
 feat[22] = different levels in X for feat 44
 feat[23] = different levels in Y for feat 44

 feat[24] = number of arc parts
 feat[25] = number of circle parts
 feat[26] = number of X-levels in symbol
 feat[27] = number of Y-levels in symbol

 feat[28] = number of linear strokes (in_rot==0)
 feat[29] = number of pos. rot. strokes (in_rot==1)
 feat[30] = number of neg. rot. strokes (in_rot==7)

 feat[31] = number of linear strokes where dir=0,1
 feat[32] = number of linear strokes where dir=2,3
 feat[33] = number of linear strokes where dir=4,5
 feat[34] = number of linear strokes where dir=6,7

 feat[35] = number of strokes where out_rot=0,1
 feat[36] = number of strokes where out_rot=2,3
 feat[37] = number of strokes where out_rot=4,5
 feat[38] = number of strokes where out_rot=6,7

 feat[39] = number of spaces where q=0,1
 feat[40] = number of spaces where q=2,3
 feat[41] = number of spaces where q=4,5
 feat[42] = number of spaces where q=6,7
```

```
    feat[43] = number of spaces where out_rot=0,1
    feat[44] = number of spaces where out_rot=2,3
    feat[45] = number of spaces where out_rot=4,5
    feat[46] = number of spaces where out_rot=6,7

    feat[47] = number of min X.
    feat[48] = number of max X.
    feat[49] = number of min Y.
    feat[50] = number of max Y.

    feat[51] = sum length of spaces.
    feat[52] = relative position of spaces.
***************************************************************** **/
```

APPENDIX C

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <values.h>
#include <mem.h>
#include <string.h>
#include <process.h>

#define GRID 15
#define MAX_NUM_OF_POINTS 1000
#define MAX_NUM_OF_STROKES 64
#define MAX_NUM_OF_EXTREMOMS 32
#define NUM_FEAT 53
#define NUM_SYMBOLS 72
#define DIVISER 6
#define TRUE  1
#define FALSE 0

typedef int boolean;

struct TABLE_STROKES_NEW {
  unsigned strk_num      ;
  int      delta_x       ;
  int      delta_y       ;
  unsigned lenght        ;
  unsigned pen_status    ;
  int rotation           ;
};

struct POINT {
  unsigned x_cor      : 10;
  unsigned y_cor      : 10;
  unsigned pen_status : 4;
  };

struct TABLE_EXTREMOMS {
  unsigned p_ndx   : 10;
  unsigned x_cor   : 10;
  unsigned y_cor   : 10;
  unsigned pen_sts : 1;
  unsigned dir     : 4;
  unsigned in_rot  : 4;
  unsigned out_rot : 4;
  unsigned susp    : 1;
  unsigned reserve : 4;
  };

struct FEATURES {
  char IDfeat[11];  /* Id feature for name */
  unsigned int feat[NUM_FEAT];
  };
```

```
/*************************************************************** ***
 feat[0]  = relative size of symbol 32*div_round(Xwidth,Ywidth)
 feat[1]  = number of points
 feat[2]  = number of strokes
 feat[3]  = number of spaces
 feat[4]  = number of different levels in X for end of space
 feat[5]  = number of different levels in Y for end of space

 feat[6]  = number of Ext X in neg.rot
 feat[7]  = different levels in X for minX
 feat[8]  = different levels in Y for minX

 feat[9]  = number of Ext in X pos.rot
 feat[10] = different levels in X for maxX
 feat[11] = different levels in Y for maxX

 feat[12] = number of Ext in Y neg.rot
 feat[13] = different levels in X for minY
 feat[14] = different levels in Y for minY

 feat[15] = number of Ext in Y pos.rot
 feat[16] = different levels in X for maxY
 feat[17] = different levels in Y for maxY

 feat[18] = number of ExtXY  for neg. rot
 feat[19] = different levels in X for feat 41
 feat[20] = different levels in Y for feat 41

 feat[21] = number of ExtXY  for pos. rot
 feat[22] = different levels in X for feat 44
 feat[23] = different levels in Y for feat 44

 feat[24] = number of arc parts
 feat[25] = number of circle parts
 feat[26] = number of X-levels in symbol
 feat[27] = number of Y-levels in symbol

 feat[28] = number of linear strokes (in_rot==0)
 feat[29] = number of pos. rot. strokes (in_rot==1)
 feat[30] = number of neg. rot. strokes (in_rot==7)

 feat[31] = number of linear strokes where dir=0,1
 feat[32] = number of linear strokes where dir=2,3
 feat[33] = number of linear strokes where dir=4,5
 feat[34] = number of linear strokes where dir=6,7

 feat[35] = number of strokes where out_rot=0,1
 feat[36] = number of strokes where out_rot=2,3
 feat[37] = number of strokes where out_rot=4,5
 feat[38] = number of strokes where out_rot=6,7

 feat[39] = number of spaces where q=0,1
 feat[40] = number of spaces where q=2,3
 feat[41] = number of spaces where q=4,5
 feat[42] = number of spaces where q=6,7
```

```
    feat[43] = number of spaces where out_rot=0,1
    feat[44] = number of spaces where out_rot=2,3
    feat[45] = number of spaces where out_rot=4,5
    feat[46] = number of spaces where out_rot=6,7

    feat[47] = number of min X.
    feat[48] = number of max X.
    feat[49] = number of min Y.
    feat[50] = number of max Y.

    feat[51] = sum length of spaces.
    feat[52] = relative position of spaces.
******************************************************************* **/
```

```
/*                  M A I N   F O R   T E A C H I N G                    */
#include "main.h"
#include "stdio.h"
void main()
{
   FILE *f_point ;

   struct FEATURES feat;
   struct FEATURES min_feat;
   struct FEATURES max_feat;

   int feat0;
   int first_feat=1;
   int file_ndx1=2,file_ndx2=0;
   char Ctemp[10];
   char Binfile[30],infile[30],minfile[30],maxfile[30];
   char SymId[11],BSymId[11];

/*    Delete all the Space's files which hold the information of
      previous teaching                                                  */

   {
     FILE *PF;
     PF=fopen ("space","w");fclose (PF);
   }
   system ("del space*.*");
   /*system ("del extrm.out");
   system ("del feat.out");*/

/*****************************************************************
   Open the files containing the X,Y,Pen of the symbol's examples
*****************************************************************/

   strcpy (infile,"\\work\\sym\\sym");
   strcpy (SymId,"sym");
   for (file_ndx1=0;file_ndx1<NUM_SYMBOLS;file_ndx1++)
   {
       itoa (file_ndx1,Ctemp,10);
       strcat (infile,Ctemp);
       strcat (SymId,Ctemp);
       strcat (infile,".");
       strcat (SymId,".");

       strcpy (Binfile,infile);
       strcpy (BSymId,SymId);

       file_ndx2=0;
       itoa (file_ndx2,Ctemp,10);
       strcat (infile,Ctemp);
       strcat (SymId,Ctemp);
       first_feat=1;
```

```
    while (transform(infile,&feat)!=0)
    {
      strcpy(feat.IDfeat,SymId);
      /*print_features (&feat,"feat.out");*/
      if (first_feat==1) {
       min_feat=feat;
       max_feat=feat;
       first_feat=0;
     } else /* update minimum and maximum feature according to feature
       of current symbol */
       calc_limits (&(min_feat),&(max_feat),feat);

        strcpy (infile,Binfile);
        strcpy (SymId,BSymId);
        file_ndx2++;
        itoa (file_ndx2,Ctemp,10);
        strcat (infile,Ctemp);
        strcat (SymId,Ctemp);
      } /* of while */
      strcpy (min_feat.IDfeat,BSymId);
      strcpy (max_feat.IDfeat,BSymId);

      if (first_feat!=1)
     /* saves the min feature and max feature of the symbol to  disk */
     save_min_max (min_feat,max_feat);

      strcpy (infile,"\\work\\sym\\sym");
      strcpy (SymId,"sym");
    } /* of for */

    exit (0);
 }
```

```
#include "main.h"

/* This procedure is to read the points of the symbol from a file , and
   fill it in the array of coordinates : arr_cor.
*/
int read_points (struct POINT arr_cor[],char *filename)
{
   FILE *fpointer;
   unsigned int x,y,p;
   unsigned int prev_x,prev_y,prev_pen;
   unsigned int i=0,in=0;

   fpointer=fopen (filename,"r+");

   prev_x=0;
   prev_y=0;
   prev_pen=0;
   while ( (fscanf(fpointer,"%d %d %d",&x,&y,&p)>0) && (p==0));
   while (fscanf(fpointer,"%d %d %d",&x,&y,&p) > 0)
   {
      if (in/DIVISER*DIVISER == in) {
        in++;
        arr_cor[i].x_cor=x;
        arr_cor[i].y_cor=y;
        arr_cor[i].pen_status=p;
        if (((abs(prev_x-x)+abs(prev_y-y))>3) || (prev_pen!=p)) {
          i++;
          prev_x=x;
          prev_y=y;
          prev_pen=p;
        }
      }
      else in++;
   }

   arr_cor[i].x_cor=arr_cor[i-1].x_cor;
   arr_cor[i].y_cor=arr_cor[i-1].y_cor;
   arr_cor[i].pen_status=0;


   fclose (fpointer);
   return (i);
}
```

```
/* This module includes general purpose procedures which are :
(1) def_quart : a funtion that recieves two numbers , and returns
    the quarter of these two numbers in the range of 0-7.
(2) div_round : a function to calculate the round number of the
    division of two integer numbers
*/

#include "main.h"

/*****************************************************************
   A funtion that recieves two numbers , and returns the quarter of
   these two numbers in the range of 0-7.
*****************************************************************/
int def_quart (x,y)
int x;
int y;
{
    if ((x>0 ) && (y==0)) return (0);
    if ((x>0 ) && (y>0) ) return (1);
    if ((x==0) && (y>0) ) return (2);
    if ((x<0 ) && (y>0) ) return (3);
    if ((x<0 ) && (y==0)) return (4);
    if ((x<0 ) && (y<0) ) return (5);
    if ((x==0) && (y<0) ) return (6);
    if ((x>0 ) && (y<0) ) return (7);
    return (15);
}

/*****************************************************************
   A function to returns the round number of the division of two
   integer numbers.
*****************************************************************/
int div_round (int a, int b)
 {
   int i=0;
   int temp1,temp2;

   temp1=a;
   temp2=b;
   a=abs(a);
   if (b==0) return (MAXINT);
    while(a>b)
    {
      if (b>0) a-=b;

   else a+=b;
      i++;
    }
     if ((a+a)>b) i++;
     if (((temp1>0) && (temp2>0)) || ((temp1<0) && (temp2<0)))
       return(i);
     else return (-i);
 }
```

```
#include "main.h"

/*****************************************************************
  calcualtes the space features of a symbol , (called from
  calc_features).
*****************************************************************/

void calc_space_feat (struct TABLE_EXTREMOMS extr[],int extrms,
                      struct FEATURES *feat)
{
   int i=0;
   boolean arr_x_levels[16];
   boolean arr_y_levels[16];
   unsigned int sum=0;

   for (i=0;i<=15;i++) { arr_x_levels[i]=FALSE;
         arr_y_levels[i]=FALSE;
       }

   for (i=0;i<extrms;i++)
   {
       if (extr[i].pen_sts==0)
       {
         (*feat).feat[51]=(*feat).feat[51]+
         (extr[i+1].p_ndx-extr[i].p_ndx);
         sum+=extr[i].p_ndx;
         ((*feat).feat[3]++);
         arr_x_levels[extr[i+1].x_cor]=TRUE;
         arr_y_levels[extr[i+1].y_cor]=TRUE;
         switch (extr[i+1].dir) {
           case 0 : (*feat).feat[39]++; break;
           case 1 : (*feat).feat[39]++; break;
           case 2 : (*feat).feat[40]++; break;
           case 3 : (*feat).feat[40]++; break;
           case 4 : (*feat).feat[41]++; break;
           case 5 : (*feat).feat[41]++; break;
           case 6 : (*feat).feat[42]++; break;
           case 7 : (*feat).feat[42]++; break;
           default: break;
         }
       }
   }
   (*feat).feat[52]=div_round(8*sum,(*feat).feat[1]);

    for (i=0;i<=15;i++) {
        if (arr_x_levels[i]==TRUE) (((*feat).feat[4])++);
        if (arr_y_levels[i]==TRUE) (((*feat).feat[5])++);
       }

}
```

```
/*****************************************************************
  calcualtes the rotation features of a symbol , (called from
  calc_features).
*****************************************************************/

void calc_rot (struct TABLE_EXTREMOMS extr[],int extrms,
               struct FEATURES *feat)
{
   int i;

    for (i=1;i<=extrms;i++) {
     switch (extr[i].in_rot) {
     case 0 :   {
       ((*feat).feat[28])++;
       switch (extr[i].dir) {
        case 0 : (*feat).feat[31]++;break;
        case 1 : (*feat).feat[31]++;break;
        case 2 : (*feat).feat[32]++;break;
        case 3 : (*feat).feat[32]++;break;
        case 4 : (*feat).feat[33]++;break;
        case 5 : (*feat).feat[33]++;break;
        case 6 : (*feat).feat[34]++;break;
        case 7 : (*feat).feat[34]++;break;
        default: break;
       }
     }
      case 1 :   ((*feat).feat[29])++; break;
      case 7 :   ((*feat).feat[30])++; break;
      default: break ;
     }

     if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
     {
       switch (extr[i].out_rot) {
         case 0 : (*feat).feat[35]++; break;
         case 1 : (*feat).feat[35]++; break;
         case 2 : (*feat).feat[36]++; break;
         case 3 : (*feat).feat[36]++; break;
         case 4 : (*feat).feat[37]++; break;
         case 5 : (*feat).feat[37]++; break;
         case 6 : (*feat).feat[38]++; break;
         case 7 : (*feat).feat[38]++; break;
         default: break;
       }
     }

     if ((extr[i-1].pen_sts+extr[i].pen_sts)==1)
     {
       switch (extr[i].out_rot)
         {
           case 0 : (*feat).feat[43]++;
           case 1 : (*feat).feat[43]++;
           case 2 : (*feat).feat[44]++;
           case 3 : (*feat).feat[44]++;
           case 4 : (*feat).feat[45]++;
```

```
            case 5 : (*feat).feat[45]++;
            case 6 : (*feat).feat[46]++;
            case 7 : (*feat).feat[46]++;
          }
        }
      }
    }

/*********************************************************************
  calcualtes the extremum type features of a symbol.
  (called from calc_features).
*********************************************************************/

void calc_ext_types (struct TABLE_EXTREMOMS extr[],int extrms,
                     struct FEATURES *feat)
{
   int i=0;
   boolean arr_x_levels[16];
   boolean arr_y_levels[16];

   for (i=0;i<=15;i++) {
     arr_x_levels[i]=FALSE;
     arr_y_levels[i]=FALSE;
   }

   for (i=1;i<extrms;i++)
     if (((extr[i].dir==3)&&((extr[i+1].dir==1) || (extr[i+1].dir==2)))
         ||
         ((extr[i].dir==7)&&((extr[i+1].dir==5) || (extr[i+1].dir==6))))
     {
       if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
          (*feat).feat[6]++;
     }
/*******************************************************************/
   for (i=0;i<=15;i++) {
     arr_x_levels[i]=FALSE;
     arr_y_levels[i]=FALSE;
   }

   for (i=1;i<extrms;i++)
     if (((extr[i].dir==5)&&((extr[i+1].dir==7) || (extr[i+1].dir==6)))
         ||
         ((extr[i].dir==3)&&((extr[i+1].dir==1) || (extr[i+1].dir==2)))
         ||
         ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
         ((extr[i].dir==3) && (extr[i+1].dir==7)))
     {
       if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
       {
         arr_x_levels[extr[i].x_cor] = TRUE;
         arr_y_levels[extr[i].y_cor] = TRUE;
       }
     }
    for (i=0;i<=15;i++) {
      if (arr_x_levels[i]==TRUE) (((*feat).feat[7])++);
      if (arr_y_levels[i]==TRUE) (((*feat).feat[8])++);
    }

/****************************************************************/
    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }

    for (i=1;i<extrms;i++)
      if (((extr[i].dir==1)&&((extr[i+1].dir==3) || (extr[i+1].dir==2)))
          ||
          ((extr[i].dir==5)&&((extr[i+1].dir==6) || (extr[i+1].dir==7))))
      {
         if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
          {
            ((*feat).feat[9]++);
            arr_x_levels[extr[i].x_cor] = TRUE;
            arr_y_levels[extr[i].y_cor] = TRUE;
          }
      }

/****************************************************************/
    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }

    for (i=1;i<extrms;i++)
      if (((extr[i].dir==7)&&((extr[i+1].dir==5) || (extr[i+1].dir==6)))
          ||
          ((extr[i].dir==1)&&((extr[i+1].dir==3) || (extr[i+1].dir==2)))
          ||
          ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
          ((extr[i].dir==7) && (extr[i+1].dir==3)))
      {
        if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
        {
          arr_x_levels[extr[i].x_cor] = TRUE;
          arr_y_levels[extr[i].y_cor] = TRUE;
        }
      }

    for (i=0;i<=15;i++) {
      if (arr_x_levels[i]==TRUE) (((*feat).feat[10])++);
      if (arr_y_levels[i]==TRUE) (((*feat).feat[11])++);
    }

/****************************************************************/
    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }
```

```
  for (i=1;i<extrms;i++)
    if (((extr[i].dir==5)&&((extr[i+1].dir==3) || (extr[i+1].dir==4)))
        ||
        ((extr[i].dir==1)&&((extr[i+1].dir==7) || (extr[i+1].dir==0))))
    {
      if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
        (*feat).feat[12]++;
    }

/*****************************************************************/
  for (i=0;i<=15;i++) {
    arr_x_levels[i]=FALSE;
    arr_y_levels[i]=FALSE;
  }

  for (i=1;i<extrms;i++)
    if (((extr[i].dir==7)&&((extr[i+1].dir==1) || (extr[i+1].dir==0)))
        ||
        ((extr[i].dir==5)&&((extr[i+1].dir==3) || (extr[i+1].dir==4)))
        ||
        ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
        ((extr[i].dir==7) && (extr[i+1].dir==3)))
    {
      if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
      {
         arr_x_levels[extr[i].x_cor] = TRUE;
         arr_y_levels[extr[i].y_cor] = TRUE;
      }
    }

  for (i=0;i<=15;i++) {
    if (arr_x_levels[i]==TRUE) (((*feat).feat[13])++);
    if (arr_y_levels[i]==TRUE) (((*feat).feat[14])++);
  }


 /*****************************************************************/
  for (i=0;i<=15;i++) {
    arr_x_levels[i]=FALSE;
    arr_y_levels[i]=FALSE;
  }

  for (i=1;i<extrms;i++)
    if (((extr[i].dir==3)&&((extr[i+1].dir==5) || (extr[i+1].dir==4)))
        ||
        ((extr[i].dir==7)&&((extr[i+1].dir==1) || (extr[i+1].dir==0))))
    {
      if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
         (*feat).feat[15]++;
    }
 /*****************************************************************/
  for (i=0;i<=15;i++) {
    arr_x_levels[i]=FALSE;
    arr_y_levels[i]=FALSE;
  }
```

```
for (i=1;i<extrms;i++)
  if (((extr[i].dir==1)&&((extr[i+1].dir==7) || (extr[i+1].dir==0)))
      ||
      ((extr[i].dir==3)&&((extr[i+1].dir==5) || (extr[i+1].dir==4)))
      ||
      ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
      ((extr[i].dir==3) && (extr[i+1].dir==7)))
  {
    if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
    {
       arr_x_levels[extr[i].x_cor] = TRUE;
       arr_y_levels[extr[i].y_cor] = TRUE;
    }
  }

for (i=0;i<=15;i++) {
  if (arr_x_levels[i]==TRUE) (((*feat).feat[16])++);
  if (arr_y_levels[i]==TRUE) (((*feat).feat[17])++);
}

/*****************************************************************/

for (i=0;i<=15;i++) {
  arr_x_levels[i]=FALSE;
  arr_y_levels[i]=FALSE;
}

for (i=1;i<extrms;i++)
  if (extr[i].out_rot==5)
  {
    if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
    {
       (*feat).feat[18]++;
       arr_x_levels[extr[i].x_cor] = TRUE;
       arr_y_levels[extr[i].y_cor] = TRUE;
    }
  }

for (i=0;i<=15;i++) {
  if (arr_x_levels[i]==TRUE) (((*feat).feat[19])++);
  if (arr_y_levels[i]==TRUE) (((*feat).feat[20])++);
}

/*****************************************************************/

for (i=0;i<=15;i++) {
  arr_x_levels[i]=FALSE;
  arr_y_levels[i]=FALSE;
}

for (i=1;i<extrms;i++)
  if ((extr[i].out_rot==3) || (extr[i].out_rot==4))
  {
    if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
    {
            (*feat).feat[21]++;
             arr_x_levels[extr[i].x_cor] = TRUE;
             arr_y_levels[extr[i].y_cor] = TRUE;
           }
         }

      for (i=0;i<=15;i++) {
        if (arr_x_levels[i]==TRUE) (((*feat).feat[22])++);
        if (arr_y_levels[i]==TRUE) (((*feat).feat[23])++);
      }
  }

/***********************************************************************/
   calcualtes the arc features of a symbol,
   (called from calc_features).
***********************************************************************/
void calc_arcs (struct TABLE_EXTREMOMS extr[],int extrms,
                struct FEATURES *feat)
{
    int i=0;
    int j=0;

    while (i<extrms)
    {
      if ((extr[i].in_rot==7) || (extr[i].in_rot==1))
      {
         ((*feat).feat[24])++;
         while ((extr[i].in_rot==7) || (extr[i].in_rot==1)) i++;
      }
      i++;
    }

    i=0;
    while (i<extrms)
    {
      if (extr[i].in_rot==7)
      {
         j=1;
         while (extr[i].in_rot==7) {j++;i++;}
      } else if (extr[i].in_rot==1)
     {
       j=1;
       while (extr[i].in_rot==1) {j++;i++;}
     }
      ((*feat).feat[25])=((*feat).feat[25])+(int)(j/4);
      i++;
    }
}
```

```
/*****************************************************************
  calculates the levels features (number of diufferent levels) of a
  symbol
  (called from calc_features).
*****************************************************************/
void calc_symbol_levels (struct TABLE_EXTREMOMS extr[],int extrms,
                         struct FEATURES *feat)
{
    int i=0;
    boolean arr_x_levels[16];
    boolean arr_y_levels[16];

   for (i=0;i<=15;i++) {
     arr_x_levels[i]=FALSE;
     arr_y_levels[i]=FALSE;
   }

   for (i=0;i<=extrms;i++)
   {
       arr_x_levels[extr[i].x_cor] = TRUE;
       arr_y_levels[extr[i].y_cor] = TRUE;
   }

   for (i=0;i<=15;i++) {
     if (arr_x_levels[i]==TRUE) (((*feat).feat[26])++);
     if (arr_y_levels[i]==TRUE) (((*feat).feat[27])++);
   }
}

/*****************************************************************
  calculates the number of minumums/maximums in X and Y.
  (called from calc_features).
*****************************************************************/
void calc_num_minmax (struct TABLE_EXTREMOMS extr[],int extrms,
                      struct FEATURES *feat)
{
   int i;

   for (i=1;i<extrms;i++)
   {
                     if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
     {
      if (((extr[i].dir==5)&&((extr[i+1].dir==7) ||
           (extr[i+1].dir==6))) ||
            ((extr[i].dir==3)&&((extr[i+1].dir==1) ||
            (extr[i+1].dir==2))) ||
            ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
            ((extr[i].dir==3) && (extr[i+1].dir==7)))
              (*feat).feat[47]++;

             if (((extr[i].dir==7)&&((extr[i+1].dir==5) ||
                 (extr[i+1].dir==6))) ||
                 ((extr[i].dir==1)&&((extr[i+1].dir==3) ||
                 (extr[i+1].dir==2))) ||
                 ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
                ((extr[i].dir==7) && (extr[i+1].dir==3)))
                  (*feat).feat[48]++;

        if (((extr[i].dir==7)&&((extr[i+1].dir==1) ||
            (extr[i+1].dir==0))) ||
            ((extr[i].dir==5)&&((extr[i+1].dir==3) ||
            (extr[i+1].dir==4))) ||
            ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
            ((extr[i].dir==7) && (extr[i+1].dir==3)))
              (*feat).feat[49]++;

        if (((extr[i].dir==1)&&((extr[i+1].dir==7) ||
            (extr[i+1].dir==0))) ||
            ((extr[i].dir==3)&&((extr[i+1].dir==5) ||
            (extr[i+1].dir==4))) ||
            ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
            ((extr[i].dir==3) && (extr[i+1].dir==7)))
             (*feat).feat[50]++;


      }
    }
}

/*********************************************************************
   calculates the features of a symbol depending on the extremums of
   that symbol , and fills in the structure variable: feat.
********************************************************************/
void calc_features (struct FEATURES *feat,int points,
                    struct TABLE_EXTREMOMS extr[],
int extrms,int feat0)
{
  int i;

  /* initialization */
  for (i=0;i<NUM_FEAT;i++) (*feat).feat[i]=0;

  (*feat).feat[0] = feat0  ;
  (*feat).feat[1] = points ;
  (*feat).feat[2] = extrms ;

  calc_space_feat (extr,extrms,feat);
  calc_rot  (extr,extrms,feat);
  calc_ext_types (extr,extrms,feat);
  calc_arcs (extr,extrms,feat);
  calc_symbol_levels (extr,extrms,feat);
  calc_num_minmax (extr,extrms,feat);
 }
```

```
/* This module is for calculating and manipulating extremums , it
   includes four procedures :

    (1) calc_extremums    : To find the extremums of the symbol , and
        the "change in pen" points.
    (2) find_max_extrm    : To find the max/min X-coordinate and the
        max/min Y-coordinate of the symbol.
    (3) analyse_extremums : A procedure to analyse the extremums found
        so far, mainly this procedure marks close extremums as
        suspecious extremums.
    (4) conv2levels       : Replaces the actual coordinates of the
        extremums in arr_extremums with the levels of these coordinates
        by doing quantization, (there are 16 levels). */

#include "main.h"

/*******************************************************************
   The input of this procedure is the array of coordinates which holds
   the coordinates of the symbol , and the number of these points , it
   fills the special of points (i.e. , extremums or change in pen) in
   the array
   arr_extremums , it returns the number of special points it found.
*******************************************************************/

int calc_extremoms (struct TABLE_EXTREMOMS arr_extremoms[],
                    struct POINT arr_cor[],int num_of_points)
{
    unsigned int indE;
    unsigned int ndx;
    unsigned pen_was_up;
    int crnt_quart,prev_quart;
    int crnt_rot,prev_rot;
    int Sx,Sy,PSx,PSy,a,b;
    int first_point=1;
    int Pndx=0;
    unsigned int LastNdx;
    int i;
    indE=0;
    ndx=0;
    pen_was_up=1;
    Sx=0;
    Sy=0;
    PSx=0;
    PSy=0;
    a=0;b=0;

    while (ndx<num_of_points)
      {
        while ((arr_cor[ndx].pen_status)==1)
        {
           if (pen_was_up==1)
           {  /* beginning extremom */
             if (first_point==1)
               {
                 int tSx,tSy;
      Sx=(arr_cor[ndx+1].x_cor-arr_cor[ndx].x_cor);
      Sy=(arr_cor[ndx+1].y_cor-arr_cor[ndx].y_cor);
      tSx=(arr_cor[ndx+2].x_cor-arr_cor[ndx+1].x_cor);
      tSy=(arr_cor[ndx+2].y_cor-arr_cor[ndx+1].y_cor);
      a=Sx*tSy-Sy*tSx;
      b=Sx*tSx+Sy*tSy;
      PSx=Sx;
      PSy=Sy;
    }
  else {
   /* for the previous ENDING extremom */
   Sx=(arr_cor[ndx].x_cor-arr_cor[Pndx].x_cor);
   Sy=(arr_cor[ndx].y_cor-arr_cor[Pndx].y_cor);
   PSx=(arr_cor[Pndx].x_cor-arr_cor[Pndx-1].x_cor);
   PSy=(arr_cor[Pndx].y_cor-arr_cor[Pndx-1].y_cor);
   a=PSx*Sy-PSy*Sx;
   b=PSx*Sx+PSy*Sy;
   arr_extremoms[indE].p_ndx=Pndx;
   arr_extremoms[indE].x_cor=arr_cor[Pndx].x_cor;
   arr_extremoms[indE].y_cor=arr_cor[Pndx].y_cor;
   arr_extremoms[indE].pen_sts=0;
   arr_extremoms[indE].dir=def_quart(PSx,PSy);
   arr_extremoms[indE].out_rot=def_quart(b,a);
   {
      int PPSx,PPSy;
      PPSx=(arr_cor[Pndx-1].x_cor-arr_cor[Pndx-2].x_cor);
      PPSy=(arr_cor[Pndx-1].y_cor-arr_cor[Pndx-2].y_cor);
      a=PPSx*PSy-PPSy*PSx;
      b=PPSx*PSx+PPSy*PSy;
      arr_extremoms[indE].in_rot=def_quart(b,a);
   }

   indE++;
   PSx=Sx;
   PSy=Sy;
   {
     int tSx,tSy;
     Sx=(arr_cor[ndx+1].x_cor-arr_cor[ndx].x_cor);
     Sy=(arr_cor[ndx+1].y_cor-arr_cor[ndx].y_cor);
     tSx=(arr_cor[ndx].x_cor-arr_cor[Pndx].x_cor);
     tSy=(arr_cor[ndx].y_cor-arr_cor[Pndx].y_cor);
     a=tSx*Sy-tSy*Sx;
     b=tSx*Sx+tSy*Sy;
   }
  }
       crnt_quart=def_quart(Sx,Sy);
       crnt_rot=def_quart(b,a);
         arr_extremoms[indE].p_ndx=ndx;
         arr_extremoms[indE].x_cor=arr_cor[ndx].x_cor;
         arr_extremoms[indE].y_cor=arr_cor[ndx].y_cor;
         arr_extremoms[indE].pen_sts=1;
         if (first_point==1) {
            arr_extremoms[indE].dir=15;
            arr_extremoms[indE].in_rot=15;
            arr_extremoms[indE].out_rot=15;
```

```
                              ndx++;
                  first_point=0;
                }
                   else {
                     arr_extremoms[indE].dir=def_quart(PSx,PSy);
                     arr_extremoms[indE].in_rot=0;
                     arr_extremoms[indE].out_rot=def_quart(b,a);
                   }
                     indE++;
                      pen_was_up=0;
                   } else {
                     crnt_quart=def_quart(Sx,Sy);
                     crnt_rot=def_quart(b,a);
                     if (( (crnt_quart!=prev_quart) ||
                   ((crnt_rot>=4) && (prev_rot<4)) ||
                   ((crnt_rot<4) && (prev_rot>=4))
                 )
                 && (prev_rot!=15))
                     { /* regular extremom */
               if ((abs(Sx)>0) && (abs(Sy)>0) || (b<0))
               {
                  if ((ndx-1-arr_extremoms[indE-1].p_ndx)>0)
                  {
                   arr_extremoms[indE].p_ndx=ndx-1;
                   arr_extremoms[indE].x_cor=arr_cor[ndx-1].x_cor;
                   arr_extremoms[indE].y_cor=arr_cor[ndx-1].y_cor;
                   arr_extremoms[indE].pen_sts=1;
                   arr_extremoms[indE].dir=prev_quart;
                   arr_extremoms[indE].in_rot=prev_rot;
                   arr_extremoms[indE].out_rot=crnt_rot;
                   indE++;
                  }
               }
          }
            }
            ndx++;

            Sx=(arr_cor[ndx].x_cor-arr_cor[ndx-1].x_cor);
            Sy=(arr_cor[ndx].y_cor-arr_cor[ndx-1].y_cor);
            a=PSx*Sy-PSy*Sx;
            b=PSx*Sx+PSy*Sy;
            PSx=Sx;
            PSy=Sy;
            prev_quart=crnt_quart;
            prev_rot=crnt_rot;
         } /* end of while (arr_cor[ndx].penstatus==1) */
         if ((pen_was_up==0) && (arr_cor[ndx-1].pen_status==1))
            { /* ending extremom */
              Pndx=ndx-1;
              pen_was_up=1;
              LastNdx=ndx-1;
            }
   ndx++;
 } /* ndx<=NumOfPoints */
```

```
    /* Add the last special point to the array */
    arr_extremoms[indE].p_ndx=LastNdx;
    arr_extremoms[indE].x_cor=arr_cor[LastNdx].x_cor;
    arr_extremoms[indE].y_cor=arr_cor[LastNdx].y_cor;
    arr_extremoms[indE].pen_sts=0;
    Sx=(arr_cor[LastNdx-1].x_cor-arr_cor[LastNdx-2].x_cor);
    Sy=(arr_cor[LastNdx-1].x_cor-arr_cor[LastNdx-2].x_cor);
    PSx=(arr_cor[LastNdx-2].x_cor-arr_cor[LastNdx-3].x_cor);
    PSy=(arr_cor[LastNdx-2].y_cor-arr_cor[LastNdx-3].y_cor);
    arr_extremoms[indE].dir=def_quart(Sx,Sy);
    a=PSx*Sy-PSy*Sx;
    b=PSx*Sx+PSy*Sy;
    arr_extremoms[indE].in_rot=def_quart(b,a);
    arr_extremoms[indE].out_rot=15;

    return (indE);
}


/**********************************************************************
   This procedure is to find the max/min X-coordinate and the max/min
   Y-coordinate of the symbol.
**********************************************************************/
void find_max_extrm (struct TABLE_EXTREMOMS arr_extremoms[],
                     int num_of_extrms,int *Xmin,int *Xmax,
                     int *Ymin,int *Ymax)
{
  unsigned int i=0;

  *Xmin=1200;
  *Xmax=0;
  *Ymin=1200;
  *Ymax=0;

  while (i<=num_of_extrms)
  {
    if (arr_extremoms[i].x_cor<*Xmin) *Xmin=arr_extremoms[i].x_cor;
    if (arr_extremoms[i].x_cor>*Xmax) *Xmax=arr_extremoms[i].x_cor;
    if (arr_extremoms[i].y_cor<*Ymin) *Ymin=arr_extremoms[i].y_cor;
    if (arr_extremoms[i].y_cor>*Ymax) *Ymax=arr_extremoms[i].y_cor;
    i++;
  }
}

/**********************************************************************
    correct dir,in_rot,out_rot  for suspicious points , called from
    analyse_extremums
**********************************************************************/
void correct_extrm (struct POINT arr_cor[],
                    struct TABLE_EXTREMOMS extrm[],
                    struct TABLE_EXTREMOMS arr_exc_extrm[],
                    int i,int Pi,int j)
{
   int Sx,Sy,PSx,PSy;
   int ndx1,ndx2;
```

```
    int a,b;
    ndx1=extrm[Pi].p_ndx;
    ndx2=extrm[i].p_ndx;
    if (Pi>0)
    {
       if (extrm[Pi-1].pen_sts==0)
       {
         PSx=arr_cor[ndx1].x_cor-arr_cor[extrm[Pi-1].p_ndx].x_cor;
         PSy=arr_cor[ndx1].y_cor-arr_cor[extrm[Pi-1].p_ndx].y_cor;
       }
       else {
             PSx=arr_cor[ndx1].x_cor-arr_cor[ndx1-1].x_cor;
             PSy=arr_cor[ndx1].y_cor-arr_cor[ndx1-1].y_cor;
            }
    } else {PSx=0;PSy=0;}
    Sx=arr_cor[ndx2].x_cor-arr_cor[ndx1].x_cor;
    Sy=arr_cor[ndx2].y_cor-arr_cor[ndx1].y_cor;
    a=PSx*Sy-PSy*Sx;
    b=PSx*Sx+PSy*Sy;
    if (arr_exc_extrm[j-1].out_rot!=15)
       arr_exc_extrm[j-1].out_rot=def_quart(b,a);

    PSx=Sx;
    PSy=Sy;
    arr_exc_extrm[j].dir=def_quart(PSx,PSy);
    arr_exc_extrm[j].in_rot=0;
    if (arr_exc_extrm[j].out_rot!=15)
    {
       if (extrm[i].pen_sts==0)
         {
             Sx=arr_cor[extrm[i+1].p_ndx].x_cor-arr_cor[ndx2].x_cor;
             Sy=arr_cor[extrm[i+1].p_ndx].y_cor-arr_cor[ndx2].y_cor;
         }
         else
         {
             Sx=arr_cor[ndx2+1].x_cor-arr_cor[ndx2].x_cor;
             Sy=arr_cor[ndx2+1].y_cor-arr_cor[ndx2].y_cor;
         }
       a=PSx*Sy-PSy*Sx;
       b=PSx*Sx+PSy*Sy;
       arr_exc_extrm[j].out_rot=def_quart(b,a);
    }
}

/*******************************************************************
   A procedure to analyse the extremums found so far , mainly this
   procedure marks close extremums as suspecious extremums.
*******************************************************************/
int analyse_extremoms (struct POINT arr_cor[],
                       struct TABLE_EXTREMOMS extrm[],
                       struct TABLE_EXTREMOMS arr_exc_extrm[],
                       int num_of_extrms)
{
    int i,j=0;
    int count=0;
```

```
        int Pi=0;
        boolean FirstP=TRUE;

        for (i=1;i<num_of_extrms;i++)
        {
          if (((extrm[i].x_cor-extrm[i+1].x_cor)<2 ||
               (extrm[i].y_cor-extrm[i+1].y_cor)<2) &&
               (extrm[i].in_rot!=extrm[i+1].in_rot) &&
               (extrm[i].dir==extrm[i+1].dir) &&
               (extrm[i].pen_sts==1) &&
               (extrm[i-1].pen_sts!=0)) extrm[i].susp=1;
            else {extrm[i].susp=0;count++;}
        }
        extrm[0].susp=0;
        extrm[i].susp=0;
        /*count++;*/

        for (i=0;i<=num_of_extrms;i++)
        {

          if (extrm[i].susp==0) {
                  arr_exc_extrm[j].p_ndx=extrm[i].p_ndx;
                  arr_exc_extrm[j].x_cor=extrm[i].x_cor;
                  arr_exc_extrm[j].y_cor=extrm[i].y_cor;
                  arr_exc_extrm[j].pen_sts=extrm[i].pen_sts;
                  arr_exc_extrm[j].dir=extrm[i].dir;
                  arr_exc_extrm[j].in_rot =extrm[i].in_rot;
                  arr_exc_extrm[j].out_rot=extrm[i].out_rot;
                  if (FirstP==FALSE)
                  {
                   correct_extrm (arr_cor,extrm,
                   arr_exc_extrm,i,Pi,j);
                   FirstP=TRUE;
                  }
           if (extrm[i].pen_sts==0) FirstP=TRUE;
           j++;
          }
  else {
  if (FirstP) Pi=i-1;
  FirstP=FALSE;
        }
       }
       {
         /*FILE *fp;
         fp=fopen ("extrm.out","ab+");
         for (i=0;i<=count;i++)
         {
    fprintf (fp,"%d  %d  %d  %d %d  %d %d \n",
     arr_exc_extrm[i].p_ndx,arr_exc_extrm[i].x_cor,
     arr_exc_extrm[i].y_cor,arr_exc_extrm[i].pen_sts,
     arr_exc_extrm[i].dir,arr_exc_extrm[i].in_rot,
     arr_exc_extrm[i].out_rot);
         }
    fprintf (fp,"****************************************\n");
         fclose (fp);*/
    }
    return (count);
 }

/**********************************************************************
   Replaces the actual coordinates of the extremums in arr_extremums
   with the levels of these coordinates by doing quantization,(there are
   16 levels).it also calculates feat0 : i.e. the relative size of
   symbol=32*div_round(Xwidth,Ywidth).
**********************************************************************/
void conv2levels (struct TABLE_EXTREMOMS arr_extremoms[],
   int num_of_extras,int *feat0)
{
   int x_width,y_width;
   int Xmin,Xmax,Ymin,Ymax;
   int i=0;

   find_max_extrm (arr_extremoms,num_of_extras,&Xmin,&Xmax,&Ymin,&Ymax);
   x_width=Xmax-Xmin;
   y_width=Ymax-Ymin;

   while (i<=num_of_extras)
   {
     arr_extremoms[i].x_cor=div_round((arr_extremoms[i].x_cor-
                             Xmin)*GRID,x_width);
     arr_extremoms[i].y_cor=div_round((arr_extremoms[i].y_cor-
                             Ymin)*GRID,y_width);
      i++;
   }

   (*feat0)=div_round(50*x_width,y_width);
}
```

```
#include "main.h"
void print_features (struct FEATURES *feat,char filename[])
{
  FILE *fp;
  int i;

  fp=fopen (filename,"ab+");

  /*fprintf (fp,"name of symbol =%s\n",(*feat).IDfeat);
  fprintf (fp,"size of symbol =%d\n",(*feat).feat0);
  fprintf (fp,"number of points =%d\n",(*feat).feat1);
  fprintf (fp,"number of str =%d\n",(*feat).feat2);
  fprintf (fp,"number of spaces =%d\n",(*feat).feat3);
  fprintf (fp,"number of diff. levels in X for end of
              space=%d\n",(*feat).feat4);
  fprintf (fp,"number of diff. levels in Y for end of
              space=%d\n",(*feat).feat5);
  fprintf (fp,"number of spaces for q<4 =%d\n",(*feat).feat6);
  fprintf (fp,"number of spaces for q>=4 =%d\n",(*feat).feat8);
  fprintf (fp,"number of spaces ro<4
                      =%d\n",(*feat).feat73);
  fprintf (fp,"number of spaces ro>=4
                      =%d\n",(*feat).feat75);

  fprintf (fp,"number of linear strokes
                    =%d\n",(*feat).feat10);
  fprintf (fp,"number of linear strokes where dir<4
                    =%d\n",(*feat).feat69);
  fprintf (fp,"number of linear strokes where dir>=4
                    =%d\n",(*feat).feat71);

  fprintf (fp,"number of strokes with pos.rot
                    =%d\n",(*feat).feat11);
  fprintf (fp,"number of strokes with neg.rot
                    =%d\n",(*feat).feat12);

  fprintf (fp,"number of strokes ro<4
                    =%d\n",(*feat).feat13);
  fprintf (fp,"number of strokes ro>=4
                    =%d\n",(*feat).feat15);

  fprintf (fp,"number of Ext in X neg. rot.
                    =%d\n",(*feat).feat17);
  fprintf (fp,"diff. levels in X for minX   =%d\n",(*feat).feat18);
  fprintf (fp,"diff. levels in Y for minX   =%d\n",(*feat).feat19);

  fprintf (fp,"number of Ext in X pos. rot. =%d\n",(*feat).feat20);
  fprintf (fp,"diff. levels in X for maxX   =%d\n",(*feat).feat21);
  fprintf (fp,"diff. levels in Y for maxX   =%d\n",(*feat).feat22);

  fprintf (fp,"number of Ext in Y neg. rot. =%d\n",(*feat).feat29);
  fprintf (fp,"diff. levels in X for minY   =%d\n",(*feat).feat30);
  fprintf (fp,"diff. levels in Y for minY   =%d\n",(*feat).feat31);

  fprintf (fp,"number of Ext in Y pos. rot. =%d\n",(*feat).feat32);
```

```
    fprintf (fp,"diff. levels in X for maxY   =%d\n",(*feat).feat33);
    fprintf (fp,"diff. levels in Y for maxY   =%d\n",(*feat).feat34);

    fprintf (fp,"number of ext XY for neg. rot=%d\n",(*feat).feat41);
    fprintf (fp,"different levels in X for ext XY-neg.rot
                                               =%d\n",(*feat).feat42);
    fprintf (fp,"different levels in Y for ext XY-neg.rot
                                               =%d\n",(*feat).feat43);

    fprintf (fp,"number of ext XY for pos. rot
                                               =%d\n",(*feat).feat44);
    fprintf (fp,"different levels in X for ext XY-pos.rot
                                               =%d\n",(*feat).feat45);
    fprintf (fp,"different levels in Y for ext XY-pos.rot
                                               =%d\n",(*feat).feat46);

    fprintf (fp,"number of arc parts          =%d\n",(*feat).feat65);
    fprintf (fp,"number of circle parts       =%d\n",(*feat).feat66);

    fprintf (fp,"number of different X-levels =%d\n",(*feat).feat67);

    fprintf (fp,"number of different Y-levels =%d\n",(*feat).feat68);


    fprintf (fp,"sum of square weights
                                        =%d\n",(*feat).sum_sqr_wgt);*/

    fprintf (fp,"\n%s\n",(*feat).IDfeat);
    for (i=0;i<NUM_FEAT;i++)
        {
     fprintf (fp,"%3d",(*feat).feat[i]);
     if ((i==20) || (i==40)) fprintf (fp,"\n");
        }

    fclose (fp);
}
```

```
#include "main.h"
int transform(char infile[30],struct FEATURES *feat)
{
   struct POINT arr_cor[MAX_NUM_OF_POINTS]; /* array of coordinates */
   struct TABLE_EXTREMOMS arr_extremoms[MAX_NUM_OF_EXTREMOMS];
      /* array of special points (i.e. extremums , changing of pen
         position */
   struct TABLE_EXTREMOMS arr_exc_extrms[MAX_NUM_OF_EXTREMOMS];
        /* array of excluded extremums , it is excluded from the array
           of extremums by excluding close points */

   int feat0;
   int num_of_points,num_of_extrms,exc_num_extrms;

/*********************************************************************
  Reading a file and calculate the number of points for the symbol,and
  stores the points read in the array : arr_cor .
*********************************************************************/
  num_of_points=read_points (arr_cor,infile);
  if (num_of_points==0) return (0);

 /********************************************************************
  Calculates the extremums from the array of coordinates , and stores
  it in the array : arr_extremums , the function returns the number of
  extremums it calculated , and it is stored in num_of_extrms.
  *******************************************************************/
  num_of_extrms=calc_extremoms (arr_extremoms,arr_cor,num_of_points);

 /******************************************************************* *
  Replaces the actual coordinates of the extremums in arr_extremums
  with the levels of these coordinates by doing quantization. (there
  are 16 levels)

*********************************************************************/
  conv2levels (arr_extremoms,num_of_extrms,&feat0);

/******************************************************************* *
Excludes close points in the extremums array and stores the remaining
extremums in arr_exc_extrms , it returns the number of extremums left
after excluding.
******************************************************************* */
exc_num_extrms=analyse_extremoms (arr_cor,arr_extremoms,
  arr_exc_extrms,num_of_extrms);


/******************************************************************* *
calculates the features of the symbol , and stores it in the structure
variable feat.

******************************************************************* */
calc_features
 (&(*feat),num_of_points,arr_exc_extrms,exc_num_extrms,feat0);

  return (num_of_points);
}
```

```
#include "main.h"
/*****************************************************************
   a procedure to update the minimum and maximum limits of a symbol ,
   according to features of entered symbol (i.e. : feat)
****************************************************************/
void calc_limits (struct FEATURES *min_feat,struct FEATURES *max_feat,
                  struct FEATURES feat)
{
 int i;

 for (i=0;i<NUM_FEAT;i++)
 {
    if ((feat.feat[i])<((*min_feat).feat[i]))
       ((*min_feat).feat[i])=(feat.feat[i]);
    if ((feat.feat[i])>((*max_feat).feat[i]))
       ((*max_feat).feat[i])=(feat.feat[i]);
 }

}

/*****************************************************************
  Saves minimum and maximum features of a symbol do disk (i.e. spaces
  files)
****************************************************************/
void save_min_max (struct FEATURES min_feat,
                   struct FEATURES max_feat)
{
   FILE *fp;
   char space[8],Ctemp[8];
   int temp;

   strcpy (space,"space");
   temp=min_feat.feat[3];
   itoa (temp,Ctemp,10);
   strcat (space,Ctemp);

   if ((fp=fopen(space,"ab+"))!=NULL)
   {
      fwrite (&min_feat,sizeof(min_feat),1,fp);
      fwrite (&max_feat,sizeof(max_feat),1,fp);
      fclose (fp);
   }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <values.h>
#include <mem.h>
#include <string.h>
#include <process.h>

#define GRID 15
#define MAX_NUM_OF_POINTS 1000
#define MAX_NUM_OF_STROKES 64
#define MAX_NUM_OF_EXTREMOMS 32
#define NUM_FEAT 53
#define NUM_SYMBOLS 72
#define DIVISER 6
#define TRUE  1
#define FALSE 0

typedef int boolean;

struct TABLE_STROKES_NEW {
  unsigned strk_num       ;
  int      delta_x        ;
  int      delta_y        ;
  unsigned lenght         ;
  unsigned pen_status     ;
  int rotation            ;
};

struct POINT {
  unsigned x_cor      : 10;
  unsigned y_cor      : 10;
  unsigned pen_status : 4;
  };

struct TABLE_EXTREMOMS {
  unsigned p_ndx   : 10;
  unsigned x_cor   : 10;
  unsigned y_cor   : 10;
  unsigned pen_sts : 1;
  unsigned dir     : 4;
  unsigned in_rot  : 4;
  unsigned out_rot : 4;
  unsigned susp    : 1;
  unsigned reserve : 4;
  };

struct FEATURES {
  char IDfeat[11];  /* Id feature for name */
  unsigned int feat[NUM_FEAT];
  };
```

```
/****************************************************************** ***
 feat[0]  = relative size of symbol 32*div_round(Xwidth,Ywidth)
 feat[1]  = number of points
 feat[2]  = number of strokes
 feat[3]  = number of spaces
 feat[4]  = number of different levels in X for end of space
 feat[5]  = number of different levels in Y for end of space

 feat[6]  = number of Ext X in neg.rot
 feat[7]  = different levels in X for minX
 feat[8]  = different levels in Y for minX

 feat[9]  = number of Ext in X pos.rot
 feat[10] = different levels in X for maxX
 feat[11] = different levels in Y for maxX

 feat[12] = number of Ext in Y neg.rot
 feat[13] = different levels in X for minY
 feat[14] = different levels in Y for minY

 feat[15] = number of Ext in Y pos.rot
 feat[16] = different levels in X for maxY
 feat[17] = different levels in Y for maxY

 feat[18] = number of ExtXY  for neg. rot
 feat[19] = different levels in X for feat 41
 feat[20] = different levels in Y for feat 41

 feat[21] = number of ExtXY  for pos. rot
 feat[22] = different levels in X for feat 44
 feat[23] = different levels in Y for feat 44

 feat[24] = number of arc parts
 feat[25] = number of circle parts
 feat[26] = number of X-levels in symbol
 feat[27] = number of Y-levels in symbol

 feat[28] = number of linear strokes (in_rot==0)
 feat[29] = number of pos. rot. strokes (in_rot==1)
 feat[30] = number of neg. rot. strokes (in_rot==7)

 feat[31] = number of linear strokes where dir=0,1
 feat[32] = number of linear strokes where dir=2,3
 feat[33] = number of linear strokes where dir=4,5
 feat[34] = number of linear strokes where dir=6,7

 feat[35] = number of strokes where out_rot=0,1
 feat[36] = number of strokes where out_rot=2,3
 feat[37] = number of strokes where out_rot=4,5
 feat[38] = number of strokes where out_rot=6,7

 feat[39] = number of spaces where q=0,1
 feat[40] = number of spaces where q=2,3
 feat[41] = number of spaces where q=4,5
 feat[42] = number of spaces where q=6,7
```

```
    feat[43] = number of spaces where out_rot=0,1
    feat[44] = number of spaces where out_rot=2,3
    feat[45] = number of spaces where out_rot=4,5
    feat[46] = number of spaces where out_rot=6,7

    feat[47] = number of min X.
    feat[48] = number of max X.
    feat[49] = number of min Y.
    feat[50] = number of max Y.

    feat[51] = sum length of spaces.
    feat[52] = relative position of spaces.
*************************************************************** **/
```

```
/*              M A I N     F O R     R E C O G N I T I O N
                ==============================================
*/

#include "main.h"
#include <stdio.h>

/*
   The program reads the file temp.ltr from the disk , and calculates
   the matching probability for every symbol to the symbol defined in
   the file temp.ltr , and then stores the vector of probabilties for
   all symbols in the file data.in .
*/

void main()
{
   FILE *fp ;

   struct FEATURES feat;
   struct FEATURES min_feat;
   struct FEATURES max_feat;

   /* Vector of probablities , for each symbol we calculate that the
      matching probability between it and the symbol defined in
      temp.ltr    */
   struct {
     char name[9];
     int  prob;
   } vector[NUM_SYMBOLS];

   int index;
   int spaces=0;
   int prob=0;
   char filename[11];
   char Sname[8];
   char Ctemp[10];
   char spacefile[11];
   char infile[11];
   int i;
   int n_sp_sym=0;

   system ("del feat.out");

   for (i=0;i<=NUM_SYMBOLS;i++)
   {
       char SymName[20];
       char Ctemp[8];

       strcpy (SymName,"sym");
       itoa (i,Ctemp,10);
       strcat (SymName,Ctemp);
       strcat (SymName,".");
       strcpy(vector[i].name,SymName);
       vector[i].prob=0;
   }
```

```
     strcpy (filename,"temp.ltr");
     strcpy (infile,filename);

     /* get features of symbol in file: infile */
     transform (infile,&feat);
     strcpy (feat.IDfeat,infile);

     spaces=feat.feat[3];
     itoa (spaces,Ctemp,10);
     strcpy (spacefile,"space");
     strcat (spacefile,Ctemp);

  /* Open space file which corresponds to the symbol represented in the
     file temp.ltr */
     if ((fp=fopen(spacefile,"rb+"))!=NULL)
     {
        /* for every symbol with the same number of spaces get minimum
           and maximum features */
        while ((fread (&(min_feat),sizeof(min_feat),1,fp))>0)
        {
                 fread (&(max_feat),sizeof(max_feat),1,fp);

                 n_sp_sym++;
    /* calcualte the probability of recognition of symbol */
    {
        strcpy(feat.IDfeat,min_feat.IDfeat);
        print_features(&feat,"feat.out");
        strcpy (feat.IDfeat,infile);
    }

    prob=rec (feat,min_feat,max_feat);
    /* fill the probability in the appropriate index in the
       vector of probabilities for all the symbols */
    for (index=0;index<=NUM_SYMBOLS;index++)
      if (strcmp(min_feat.IDfeat,vector[index].name) == 0) {
        vector[index].prob = prob ;
        break;
      }
       }
       fclose (fp);
    }


    /* Save vector of probabilities to disk */
    fp=fopen ("data.in","w+");
    for (index=0;index<NUM_SYMBOLS;index++)
       fprintf (fp,"%d    %d\n",index,vector[index].prob);
    fclose (fp);


    exit (0);
}
```

```
#include "main.h"

/* This procedure is to read the points of the symbol from a file , and
   fill it in the array of coordinates : arr_cor.
*/
int read_points (struct POINT arr_cor[],char *filename)
{
   FILE *fpointer;
   unsigned int x,y,p;
   unsigned int prev_x,prev_y,prev_pen;
   unsigned int i=0,in=0;

   fpointer=fopen (filename,"r+");

   prev_x=0;
   prev_y=0;
   prev_pen=0;
   while ( (fscanf(fpointer,"%d %d %d",&x,&y,&p)>0) && (p==0));
   while (fscanf(fpointer,"%d %d %d",&x,&y,&p) > 0)
   {
      if (in/DIVISER*DIVISER == in) {
        in++;
        arr_cor[i].x_cor=x;
        arr_cor[i].y_cor=y;
        arr_cor[i].pen_status=p;
        if (((abs(prev_x-x)+abs(prev_y-y))>3) || (prev_pen!=p)) {
          i++;
          prev_x=x;
          prev_y=y;
          prev_pen=p;
        }
      }
      else in++;
   }

   arr_cor[i].x_cor=arr_cor[i-1].x_cor;
   arr_cor[i].y_cor=arr_cor[i-1].y_cor;
   arr_cor[i].pen_status=0;


   fclose (fpointer);
   return (i);
}
```

```
/* This module includes general purpose procedures which are :
(1) def_quart : a funtion that recieves two numbers , and returns
    the quarter of these two numbers in the range of 0-7.
(2) div_round : a function to calculate the round number of the
    division of two integer numbers
*/

#include "main.h"

/********************************************************************
   A funtion that recieves two numbers , and returns the quarter of
   these two numbers in the range of 0-7.
********************************************************************/
int def_quart (x,y)
int x;
int y;
{
    if ((x>0 ) && (y==0)) return (0);
    if ((x>0 ) && (y>0) ) return (1);
    if ((x==0) && (y>0) ) return (2);
    if ((x<0 ) && (y>0) ) return (3);
    if ((x<0 ) && (y==0)) return (4);
    if ((x<0 ) && (y<0) ) return (5);
    if ((x==0) && (y<0) ) return (6);
    if ((x>0 ) && (y<0) ) return (7);
    return (15);
}

/********************************************************************
   A function to returns the round number of the division of two
   integer numbers.
********************************************************************/
int div_round (int a, int b)
 {
   int i=0;
   int temp1,temp2;

   temp1=a;
   temp2=b;
   a=abs(a);
   if (b==0) return (MAXINT);
    while(a>b)
    {
      if (b>0) a-=b;

   else a+=b;
      i++;
    }
     if ((a+a)>b) i++;
     if (((temp1>0) && (temp2>0)) || ((temp1<0) && (temp2<0)))
       return(i);
     else return (-i);
 }
```

```
#include "main.h"

/*********************************************************************
  calcualtes the space features of a symbol . (called from
  calc_features).
*********************************************************************/

void calc_space_feat (struct TABLE_EXTREMOMS extr[],int extrms,
                      struct FEATURES *feat)
{
   int i=0;
   boolean arr_x_levels[16];
   boolean arr_y_levels[16];
   unsigned int sum=0;

   for (i=0;i<=15;i++) { arr_x_levels[i]=FALSE;
            arr_y_levels[i]=FALSE;
      }

   for (i=0;i<extrms;i++)
   {
       if (extr[i].pen_sts==0)
       {
         (*feat).feat[51]=(*feat).feat[51]+
         (extr[i+1].p_ndx-extr[i].p_ndx);
         sum+=extr[i].p_ndx;
         ((*feat).feat[3]++);
         arr_x_levels[extr[i+1].x_cor]=TRUE;
         arr_y_levels[extr[i+1].y_cor]=TRUE;
         switch (extr[i+1].dir) {
           case 0 : (*feat).feat[39]++; break;
           case 1 : (*feat).feat[39]++; break;
           case 2 : (*feat).feat[40]++; break;
           case 3 : (*feat).feat[40]++; break;
           case 4 : (*feat).feat[41]++; break;
           case 5 : (*feat).feat[41]++; break;
           case 6 : (*feat).feat[42]++; break;
           case 7 : (*feat).feat[42]++; break;
           default: break;
         }
       }
   }
   (*feat).feat[52]=div_round(8*sum,(*feat).feat[1]);

    for (i=0;i<=15;i++) {
        if (arr_x_levels[i]==TRUE) (((*feat).feat[4])++);
        if (arr_y_levels[i]==TRUE) (((*feat).feat[5])++);
       }

}
```

```
/**********************************************************************
  calcualtes the rotation features of a symbol . (called from
  calc_features).
**********************************************************************/

void calc_rot (struct TABLE_EXTREMOMS extr[],int extrms,
               struct FEATURES *feat)
{
   int i;

    for (i=1;i<=extrms;i++) {
     switch (extr[i].in_rot) {
     case 0 :    {
      ((*feat).feat[28])++;
       switch (extr[i].dir) {
        case 0 : (*feat).feat[31]++;break;
        case 1 : (*feat).feat[31]++;break;
        case 2 : (*feat).feat[32]++;break;
        case 3 : (*feat).feat[32]++;break;
        case 4 : (*feat).feat[33]++;break;
        case 5 : (*feat).feat[33]++;break;
        case 6 : (*feat).feat[34]++;break;
        case 7 : (*feat).feat[34]++;break;
        default: break;
       }
    }
     case 1 :   ((*feat).feat[29])++;   eak;
     case 7 :   ((*feat).feat[30])++; break;
     default: break ;
    }

    if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
    {
      switch (extr[i].out_rot) {
        case 0 : (*feat).feat[35]++; break;
        case 1 : (*feat).feat[35]++; break;
        case 2 : (*feat).feat[36]++; break;
        case 3 : (*feat).feat[36]++; break;
        case 4 : (*feat).feat[37]++; break;
        case 5 : (*feat).feat[37]++; break;
        case 6 : (*feat).feat[38]++; break;
        case 7 : (*feat).feat[38]++; break;
        default: break;
      }
    }

    if ((extr[i-1].pen_sts+extr[i].pen_sts)==1)
    {
      switch (extr[i].out_rot)
        {
          case 0 : (*feat).feat[43]++;
          case 1 : (*feat).feat[43]++;
          case 2 : (*feat).feat[44]++;
          case 3 : (*feat).feat[44]++;
          case 4 : (*feat).feat[45]++;
```

```
            case 5 : (*feat).feat[45]++;
            case 6 : (*feat).feat[46]++;
            case 7 : (*feat).feat[46]++;
          }
        }
      }
    }

/*******************************************************************
  calcualtes the extremum type features of a symbol,
  (called from calc_features).
*******************************************************************/

void calc_ext_types (struct TABLE_EXTREMOMS extr[],int extrms,
                     struct FEATURES *feat)
{
   int i=0;
   boolean arr_x_levels[16];
   boolean arr_y_levels[16];

   for (i=0;i<=15;i++) {
     arr_x_levels[i]=FALSE;
     arr_y_levels[i]=FALSE;
   }

   for (i=1;i<extrms;i++)
     if (((extr[i].dir==3)&&((extr[i+1].dir==1) || (extr[i+1].dir==2)))
          ||
         ((extr[i].dir==7)&&((extr[i+1].dir==5) || (extr[i+1].dir==6))))
     {
       if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
          (*feat).feat[6]++;
     }

/****************************************************************/
   for (i=0;i<=15;i++) {
     arr_x_levels[i]=FALSE;
     arr_y_levels[i]=FALSE;
   }

   for (i=1;i<extrms;i++)
     if (((extr[i].dir==5)&&((extr[i+1].dir==7) || (extr[i+1].dir==6)))
          ||
         ((extr[i].dir==3)&&((extr[i+1].dir==1) || (extr[i+1].dir==2)))
          ||
         ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
         ((extr[i].dir==3) && (extr[i+1].dir==7)))
     {
       if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
       {
         arr_x_levels[extr[i].x_cor] = TRUE;
         arr_y_levels[extr[i].y_cor] = TRUE;
       }
     }
  for (i=0;i<=15;i++) {
    if (arr_x_levels[i]==TRUE) (((*feat).feat[7])++);
    if (arr_y_levels[i]==TRUE) (((*feat).feat[8])++);
  }

/*****************************************************************/
  for (i=0;i<=15;i++) {
    arr_x_levels[i]=FALSE;
    arr_y_levels[i]=FALSE;
  }

  for (i=1;i<extrms;i++)
    if (((extr[i].dir==1)&&((extr[i+1].dir==3) || (extr[i+1].dir==2)))
       ||
       ((extr[i].dir==5)&&((extr[i+1].dir==6) || (extr[i+1].dir==7))))
    {
       if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
        {
          ((*feat).feat[9]++);
          arr_x_levels[extr[i].x_cor] = TRUE;
          arr_y_levels[extr[i].y_cor] = TRUE;
        }
    }

/*****************************************************************/
  for (i=0;i<=15;i++) {
    arr_x_levels[i]=FALSE;
    arr_y_levels[i]=FALSE;
  }

  for (i=1;i<extrms;i++)
    if (((extr[i].dir==7)&&((extr[i+1].dir==5) || (extr[i+1].dir==6)))
       ||
       ((extr[i].dir==1)&&((extr[i+1].dir==3) || (extr[i+1].dir==2)))
       ||
       ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
       ((extr[i].dir==7) && (extr[i+1].dir==3)))
    {
      if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
      {
        arr_x_levels[extr[i].x_cor] = TRUE;
        arr_y_levels[extr[i].y_cor] = TRUE;
      }
    }

  for (i=0;i<=15;i++) {
    if (arr_x_levels[i]==TRUE) (((*feat).feat[10])++);
    if (arr_y_levels[i]==TRUE) (((*feat).feat[11])++);
  }

/*****************************************************************/
  for (i=0;i<=15;i++) {
    arr_x_levels[i]=FALSE;
    arr_y_levels[i]=FALSE;
  }
```

```
    for (i=1;i<extrms;i++)
      if (((extr[i].dir==5)&&((extr[i+1].dir==3) || (extr[i+1].dir==4)))
          ||
          ((extr[i].dir==1)&&((extr[i+1].dir==7) || (extr[i+1].dir==0))))
      {
        if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
          (*feat).feat[12]++;
      }
/*****************************************************************/
    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }

    for (i=1;i<extrms;i++)
      if (((extr[i].dir==7)&&((extr[i+1].dir==1) || (extr[i+1].dir==0)))
          ||
          ((extr[i].dir==5)&&((extr[i+1].dir==3) || (extr[i+1].dir==4)))
          ||
          ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
          ((extr[i].dir==7) && (extr[i+1].dir==3)))
      {
        if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
        {
            arr_x_levels[extr[i].x_cor] = TRUE;
            arr_y_levels[extr[i].y_cor] = TRUE;
        }
      }

    for (i=0;i<=15;i++) {
      if (arr_x_levels[i]==TRUE) (((*feat).feat[13])++);
      if (arr_y_levels[i]==TRUE) (((*feat).feat[14])++);
    }


/*****************************************************************/
    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }

    for (i=1;i<extrms;i++)
      if (((extr[i].dir==3)&&((extr[i+1].dir==5) || (extr[i+1].dir==4)))
          ||
          ((extr[i].dir==7)&&((extr[i+1].dir==1) || (extr[i+1].dir==0))))
      {
        if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
          (*feat).feat[15]++;
      }
/*****************************************************************/
    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }
```

```
 for (i=1;i<extrms;i++)
   if (((extr[i].dir==1)&&((extr[i+1].dir==7) || (extr[i+1].dir==0)))
      ||
      ((extr[i].dir==3)&&((extr[i+1].dir==5) || (extr[i+1].dir==4)))
      ||
      ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
      ((extr[i].dir==3) && (extr[i+1].dir==7)))
   {
     if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
     {
        arr_x_levels[extr[i].x_cor] = TRUE;
        arr_y_levels[extr[i].y_cor] = TRUE;
     }
   .}

 for (i=0;i<=15;i++) {
   if (arr_x_levels[i]==TRUE) (((*feat).feat[16])++);
   if (arr_y_levels[i]==TRUE) (((*feat).feat[17])++);
 }

/*****************************************************************/

 for (i=0;i<=15;i++) {
   arr_x_levels[i]=FALSE;
   arr_y_levels[i]=FALSE;
 }

 for (i=1;i<extrms;i++)
   if (extr[i].out_rot==5)
   {
     if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
     {
        (*feat).feat[18]++;
        arr_x_levels[extr[i].x_cor] = TRUE;
        arr_y_levels[extr[i].y_cor] = TRUE;
     }
   }

 for (i=0;i<=15;i++) {
   if (arr_x_levels[i]==TRUE) (((*feat).feat[19])++);
   if (arr_y_levels[i]==TRUE) (((*feat).feat[20])++);
 }

/*****************************************************************/

 for (i=0;i<=15;i++) {
   arr_x_levels[i]=FALSE;
   arr_y_levels[i]=FALSE;
 }

 for (i=1;i<extrms;i++)
   if ((extr[i].out_rot==3) || (extr[i].out_rot==4))
   {
     if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
     {
          (*feat).feat[21]++;
           arr_x_levels[extr[i].x_cor] = TRUE;
           arr_y_levels[extr[i].y_cor] = TRUE;
         }
       }

    for (i=0;i<=15;i++) {
      if (arr_x_levels[i]==TRUE) (((*feat).feat[22])++);
      if (arr_y_levels[i]==TRUE) (((*feat).feat[23])++);
    }
}


/*************************************************************************/
  calcualtes the arc features of a symbol,
  (called from calc_features).
*************************************************************************/
void calc_arcs (struct TABLE_EXTREMOMS extr[],int extrms,
                struct FEATURES *feat)
{
   int i=0;
   int j=0;

   while (i<extrms)
   {
     if ((extr[i].in_rot==7) || (extr[i].in_rot==1))
     {
        ((*feat).feat[24])++;
        while ((extr[i].in_rot==7) || (extr[i].in_rot==1)) i++;
     }
     i++;
   }

   i=0;
   while (i<extrms)
   {
     if (extr[i].in_rot==7)
     {
        j=1;
        while (extr[i].in_rot==7) {j++;i++;}
     } else if (extr[i].in_rot==1)
    {
      j=1;
      while (extr[i].in_rot==1) {j++;i++;}
    }
     ((*feat).feat[25])=((*feat).feat[25])+(int)(j/4);
     i++;
   }
}
```

```
/*****************************************************************
  calculates the levels features (number of diufferent levels) of a
  symbol
  (called from calc_features).
*****************************************************************/
void calc_symbol_levels (struct TABLE_EXTREMOMS extr[],int extrms,
                         struct FEATURES *feat)
{
     int i=0;
     boolean arr_x_levels[16];
     boolean arr_y_levels[16];

    for (i=0;i<=15;i++) {
      arr_x_levels[i]=FALSE;
      arr_y_levels[i]=FALSE;
    }

    for (i=0;i<=extrms;i++)
    {
        arr_x_levels[extr[i].x_cor] = TRUE;
        arr_y_levels[extr[i].y_cor] = TRUE;
    }

    for (i=0;i<=15;i++) {
      if (arr_x_levels[i]==TRUE) (((*feat).feat[26])++);
      if (arr_y_levels[i]==TRUE) (((*feat).feat[27])++);
    }
}

/*****************************************************************
  calculates the number of minumums/maximums in X and Y.
  (called from calc_features).
*****************************************************************/
void calc_num_minmax (struct TABLE_EXTREMOMS extr[],int extrms,
                      struct FEATURES *feat)
{
   int i;

   for (i=1;i<extrms;i++)
   {
                     if ((extr[i-1].pen_sts+extr[i].pen_sts)==2)
     {
      if (((extr[i].dir==5)&&((extr[i+1].dir==7) ||
           (extr[i+1].dir==6))) ||
            ((extr[i].dir==3)&&((extr[i+1].dir==1) ||
            (extr[i+1].dir==2))) ||
            ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
            ((extr[i].dir==3) && (extr[i+1].dir==7)))
              (*feat).feat[47]++;

           if (((extr[i].dir==7)&&((extr[i+1].dir==5) ||
                (extr[i+1].dir==6))) ||
                ((extr[i].dir==1)&&((extr[i+1].dir==3) ||
                (extr[i+1].dir==2))) ||
                ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
                ((extr[i].dir==7) && (extr[i+1].dir==3)))
                  (*feat).feat[48]++;

          if (((extr[i].dir==7)&&((extr[i+1].dir==1) ||
              (extr[i+1].dir==0))) ||
              ((extr[i].dir==5)&&((extr[i+1].dir==3) ||
              (extr[i+1].dir==4))) ||
              ((extr[i].dir==5) && (extr[i+1].dir==1)) ||
              ((extr[i].dir==7) && (extr[i+1].dir==3)))
                (*feat).feat[49]++;

          if (((extr[i].dir==1)&&((extr[i+1].dir==7) ||
              (extr[i+1].dir==0))) ||
              ((extr[i].dir==3)&&((extr[i+1].dir==5) ||
              (extr[i+1].dir==4))) ||
              ((extr[i].dir==1) && (extr[i+1].dir==5)) ||
              ((extr[i].dir==3) && (extr[i+1].dir==7)))
               (*feat).feat[50]++;


      }
    }
}

/**********************************************************************
   calculates the features of a symbol depending on the extremums of
   that symbol , and fills in the structure variable: feat.
*********************************************************************/
void calc_features (struct FEATURES *feat,int points,
                    struct TABLE_EXTREMOMS extr[],
int extrms,int feat0)
{
  int i;

  /* initialization */
  for (i=0;i<NUM_FEAT;i++) (*feat).feat[i]=0;

  (*feat).feat[0] = feat0  ;
  (*feat).feat[1] = points ;
  (*feat).feat[2] = extrms ;

  calc_space_feat (extr,extrms,feat);
  calc_rot  (extr,extrms,feat);
  calc_ext_types (extr,extrms,feat);
  calc_arcs (extr,extrms,feat);
  calc_symbol_levels (extr,extrms,feat);
  calc_num_minmax (extr,extrms,feat);
}
```

```
/* This module is for calculating and manipulating extremums , it
   includes four procedures :

   (1) calc_extremums     : To find the extremums of the symbol , and
       the "change in pen" points.
   (2) find_max_extrm     : To find the max/min X-coordinate and the
       max/min Y-coordinate of the symbol.
   (3) analyse_extremums : A procedure to analyse the extremums found
       so far, mainly this procedure marks close extremums as
       suspecious extremums.
   (4) conv2levels        : Replaces the actual coordinates of the
       extremums in arr_extremums with the levels of these coordinates
       by doing quantization, (there are 16 levels). */

#include "main.h"

/*********************************************************************
   The input of this procedure is the array of coordinates which holds
   the coordinates of the symbol , and the number of these points , it
   fills the special of points (i.e. , extremums or change in pen) in
   the array
   arr_extremums , it returns the number of special points it found.
*********************************************************************/

int calc_extremoms (struct TABLE_EXTREMOMS arr_extremoms[],
                    struct POINT arr_cor[],int num_of_points)
{
   unsigned int indE;
   unsigned int ndx;
   unsigned pen_was_up;
   int crnt_quart,prev_quart;
   int crnt_rot,prev_rot;
   int Sx,Sy,PSx,PSy,a,b;
   int first_point=1;
   int Pndx=0;
   unsigned int LastNdx;
   int i;
   indE=0;
   ndx=0;
   pen_was_up=1;
   Sx=0;
   Sy=0;
   PSx=0;
   PSy=0;
   a=0;b=0;

   while (ndx<num_of_points)
     {
      while ((arr_cor[ndx].pen_status)==1)
      {
         if (pen_was_up==1)
         {  /* beginning extremom */
           if (first_point==1)
             {
               int tSx,tSy;
      Sx=(arr_cor[ndx+1].x_cor-arr_cor[ndx].x_cor);
      Sy=(arr_cor[ndx+1].y_cor-arr_cor[ndx].y_cor);
      tSx=(arr_cor[ndx+2].x_cor-arr_cor[ndx+1].x_cor);
      tSy=(arr_cor[ndx+2].y_cor-arr_cor[ndx+1].y_cor);
      a=Sx*tSy-Sy*tSx;
      b=Sx*tSx+Sy*tSy;
      PSx=Sx;
      PSy=Sy;
    }
  else {
   /* for the previous ENDING extremom */
   Sx=(arr_cor[ndx].x_cor-arr_cor[Pndx].x_cor);
   Sy=(arr_cor[ndx].y_cor-arr_cor[Pndx].y_cor);
   PSx=(arr_cor[Pndx].x_cor-arr_cor[Pndx-1].x_cor);
   PSy=(arr_cor[Pndx].y_cor-arr_cor[Pndx-1].y_cor);
   a=PSx*Sy-PSy*Sx;
   b=PSx*Sx+PSy*Sy;
   arr_extremoms[indE].p_ndx=Pndx;
   arr_extremoms[indE].x_cor=arr_cor[Pndx].x_cor;
   arr_extremoms[indE].y_cor=arr_cor[Pndx].y_cor;
   arr_extremoms[indE].pen_sts=0;
   arr_extremoms[indE].dir=def_quart(PSx,PSy);
   arr_extremoms[indE].out_rot=def_quart(b,a);
   {
      int PPSx,PPSy;
      PPSx=(arr_cor[Pndx-1].x_cor-arr_cor[Pndx-2].x_cor);
      PPSy=(arr_cor[Pndx-1].y_cor-arr_cor[Pndx-2].y_cor);
      a=PPSx*PSy-PPSy*PSx;
      b=PPSx*PSx+PPSy*PSy;
      arr_extremoms[indE].in_rot=def_quart(b,a);
   }

   indE++;
   PSx=Sx;
   PSy=Sy;
   {
     int tSx,tSy;
     Sx=(arr_cor[ndx+1].x_cor-arr_cor[ndx].x_cor);
     Sy=(arr_cor[ndx+1].y_cor-arr_cor[ndx].y_cor);
     tSx=(arr_cor[ndx].x_cor-arr_cor[Pndx].x_cor);
     tSy=(arr_cor[ndx].y_cor-arr_cor[Pndx].y_cor);
     a=tSx*Sy-tSy*Sx;
     b=tSx*Sx+tSy*Sy;
   }
 }
      crnt_quart=def_quart(Sx,Sy);
      crnt_rot=def_quart(b,a);
        arr_extremoms[indE].p_ndx=ndx;
        arr_extremoms[indE].x_cor=arr_cor[ndx].x_cor;
        arr_extremoms[indE].y_cor=arr_cor[ndx].y_cor;
        arr_extremoms[indE].pen_sts=1;
        if (first_point==1) {
           arr_extremoms[indE].dir=15;
           arr_extremoms[indE].in_rot=15;
           arr_extremoms[indE].out_rot=15;
```

```
                    ndx++;
          first_point=0;
        }
            else {
             arr_extremoms[indE].dir=def_quart(PSx,PSy);
             arr_extremoms[indE].in_rot=0;
             arr_extremoms[indE].out_rot=def_quart(b,a);
           }
             indE++;
              pen_was_up=0;
           } else {
             crnt_quart=def_quart(Sx,Sy);
             crnt_rot=def_quart(b,a);
             if (( (crnt_quart!=prev_quart) ||
          ((crnt_rot>=4) && (prev_rot<4)) ||
          ((crnt_rot<4) && (prev_rot>=4))
        )
        && (prev_rot!=15))
             { /* regular extremom */
      if ((abs(Sx)>0) && (abs(Sy)>0) || (b<0))
      {
         if ((ndx-1-arr_extremoms[indE-1].p_ndx)>0)
         {
          arr_extremoms[indE].p_ndx=ndx-1;
          arr_extremoms[indE].x_cor=arr_cor[ndx-1].x_cor;
          arr_extremoms[indE].y_cor=arr_cor[ndx-1].y_cor;
          arr_extremoms[indE].pen_sts=1;
          arr_extremoms[indE].dir=prev_quart;
          arr_extremoms[indE].in_rot=prev_rot;
          arr_extremoms[indE].out_rot=crnt_rot;
          indE++;
         }
      }
  }
    }
    ndx++;

    Sx=(arr_cor[ndx].x_cor-arr_cor[ndx-1].x_cor);
    Sy=(arr_cor[ndx].y_cor-arr_cor[ndx-1].y_cor);
    a=PSx*Sy-PSy*Sx;
    b=PSx*Sx+PSy*Sy;
    PSx=Sx;
    PSy=Sy;
    prev_quart=crnt_quart;
    prev_rot=crnt_rot;
  } /* end of while (arr_cor[ndx].penstatus==1) */
  if ((pen_was_up==0) && (arr_cor[ndx-1].pen_status==1))
    { /* ending extremom */
      Pndx=ndx-1;
      pen_was_up=1;
      LastNdx=ndx-1;
    }
 ndx++;
} /* ndx<=NumOfPoints */
```

```
    /* Add the last special point to the array */
    arr_extremoms[indE].p_ndx=LastNdx;
    arr_extremoms[indE].x_cor=arr_cor[LastNdx].x_cor;
    arr_extremoms[indE].y_cor=arr_cor[LastNdx].y_cor;
    arr_extremoms[indE].pen_sts=0;
    Sx=(arr_cor[LastNdx-1].x_cor-arr_cor[LastNdx-2].x_cor);
    Sy=(arr_cor[LastNdx-1].x_cor-arr_cor[LastNdx-2].x_cor);
    PSx=(arr_cor[LastNdx-2].x_cor-arr_cor[LastNdx-3].x_cor);
    PSy=(arr_cor[LastNdx-2].y_cor-arr_cor[LastNdx-3].y_cor);
    arr_extremoms[indE].dir=def_quart(Sx,Sy);
    a=PSx*Sy-PSy*Sx;
    b=PSx*Sx+PSy*Sy;
    arr_extremoms[indE].in_rot=def_quart(b,a);
    arr_extremoms[indE].out_rot=15;

    return (indE);
}

/*********************************************************************
   This procedure is to find the max/min X-coordinate and the max/min
   Y-coordinate of the symbol.
*********************************************************************/
void find_max_extrm (struct TABLE_EXTREMOMS arr_extremoms[],
                     int num_of_extrms,int *Xmin,int *Xmax,
                     int *Ymin,int *Ymax)
{
  unsigned int i=0;

  *Xmin=1200;
  *Xmax=0;
  *Ymin=1200;
  *Ymax=0;

  while (i<=num_of_extrms)
  {
    if (arr_extremoms[i].x_cor<*Xmin) *Xmin=arr_extremoms[i].x_cor;
    if (arr_extremoms[i].x_cor>*Xmax) *Xmax=arr_extremoms[i].x_cor;
    if (arr_extremoms[i].y_cor<*Ymin) *Ymin=arr_extremoms[i].y_cor;
    if (arr_extremoms[i].y_cor>*Ymax) *Ymax=arr_extremoms[i].y_cor;
    i++;
  }
}

/*********************************************************************
    correct dir,in_rot,out_rot  for suspicious points , called from
    analyse_extremums
*********************************************************************/
void correct_extrm (struct POINT arr_cor[],
                    struct TABLE_EXTREMOMS extrm[],
                    struct TABLE_EXTREMOMS arr_exc_extrm[],
                    int i,int Pi,int j)
{
   int Sx,Sy,PSx,PSy;
   int ndx1,ndx2;
```

```
    int a,b;
    ndx1=extrm[Pi].p_ndx;
    ndx2=extrm[i].p_ndx;
    if (Pi>0)
    {
       if (extrm[Pi-1].pen_sts==0)
       {
         PSx=arr_cor[ndx1].x_cor-arr_cor[extrm[Pi-1].p_ndx].x_cor;
         PSy=arr_cor[ndx1].y_cor-arr_cor[extrm[Pi-1].p_ndx].y_cor;
       }
       else {
             PSx=arr_cor[ndx1].x_cor-arr_cor[ndx1-1].x_cor;
             PSy=arr_cor[ndx1].y_cor-arr_cor[ndx1-1].y_cor;
           }
   } else {PSx=0;PSy=0;}
   Sx=arr_cor[ndx2].x_cor-arr_cor[ndx1].x_cor;
   Sy=arr_cor[ndx2].y_cor-arr_cor[ndx1].y_cor;
   a=PSx*Sy-PSy*Sx;
   b=PSx*Sx+PSy*Sy;
   if (arr_exc_extrm[j-1].out_rot!=15)
      arr_exc_extrm[j-1].out_rot=def_quart(b,a);

   PSx=Sx;
   PSy=Sy;
   arr_exc_extrm[j].dir=def_quart(PSx,PSy);
   arr_exc_extrm[j].in_rot=0;
   if (arr_exc_extrm[j].out_rot!=15)
   {
      if (extrm[i].pen_sts==0)
        {
            Sx=arr_cor[extrm[i+1].p_ndx].x_cor-arr_cor[ndx2].x_cor;
            Sy=arr_cor[extrm[i+1].p_ndx].y_cor-arr_cor[ndx2].y_cor;
        }
        else
        {
            Sx=arr_cor[ndx2+1].x_cor-arr_cor[ndx2].x_cor;
            Sy=arr_cor[ndx2+1].y_cor-arr_cor[ndx2].y_cor;
        }
      a=PSx*Sy-PSy*Sx;
      b=PSx*Sx+PSy*Sy;
      arr_exc_extrm[j].out_rot=def_quart(b,a);
   }
}

/*******************************************************************
   A procedure to analyse the extremums found so far , mainly this
   procedure marks close extremums as suspecious extremums.
*******************************************************************/
int analyse_extremoms (struct POINT arr_cor[],
                       struct TABLE_EXTREMOMS extrm[],
                       struct TABLE_EXTREMOMS arr_exc_extrm[],
                       int num_of_extrms)
{
    int i,j=0;
    int count=0;
```

```
        int Pi=0;
        boolean FirstP=TRUE;

        for (i=1;i<num_of_extrms;i++)
        {
          if (((extrm[i].x_cor-extrm[i+1].x_cor)<2 ||
               (extrm[i].y_cor-extrm[i+1].y_cor)<2) &&
               (extrm[i].in_rot!=extrm[i+1].in_rot) &&
               (extrm[i].dir==extrm[i+1].dir) &&
               (extrm[i].pen_sts==1) &&
               (extrm[i-1].pen_sts!=0)) extrm[i].susp=1;
            else {extrm[i].susp=0;count++;}
        }
       extrm[0].susp=0;
       extrm[i].susp=0;
       /*count++;*/

       for (i=0;i<=num_of_extrms;i++)
       {

         if (extrm[i].susp==0) {
                arr_exc_extrm[j].p_ndx=extrm[i].p_ndx;
                arr_exc_extrm[j].x_cor=extrm[i].x_cor;
                arr_exc_extrm[j].y_cor=extrm[i].y_cor;
                arr_exc_extrm[j].pen_sts=extrm[i].pen_sts;
                arr_exc_extrm[j].dir=extrm[i].dir;
                arr_exc_extrm[j].in_rot =extrm[i].in_rot;
                arr_exc_extrm[j].out_rot=extrm[i].out_rot;
                if (FirstP==FALSE)
                {
                 correct_extrm (arr_cor,extrm,
                 arr_exc_extrm,i,Pi,j);
                 FirstP=TRUE;
                }
         if (extrm[i].pen_sts==0) FirstP=TRUE;
         j++;
        }
   else {
   if (FirstP) Pi=i-1;
   FirstP=FALSE;
       }
      }
      {
        /*FILE *fp;
        fp=fopen ("extrm.out","ab+");
        for (i=0;i<=count;i++)
        {
    fprintf (fp,"%d  %d  %d  %d %d  %d %d \n",
     arr_exc_extrm[i].p_ndx,arr_exc_extrm[i].x_cor,
     arr_exc_extrm[i].y_cor,arr_exc_extrm[i].pen_sts,
     arr_exc_extrm[i].dir,arr_exc_extrm[i].in_rot,
     arr_exc_extrm[i].out_rot);
        }
    fprintf (fp,"*****************************************\n");
        fclose (fp);*/
    }
    return (count);
}

/**********************************************************************
  Replaces the actual coordinates of the extremums in arr_extremums
  with the levels of these coordinates by doing quantization.(there are
  16 levels).it also calculates feat0 : i.e. the relative size of
  symbol=32*div_round(Xwidth,Ywidth).
**********************************************************************/
void conv2levels (struct TABLE_EXTREMOMS arr_extremoms[],
  int num_of_extrms,int *feat0)
{
  int x_width,y_width;
  int Xmin,Xmax,Ymin,Ymax;
  int i=0;

  find_max_extrm (arr_extremoms,num_of_extrms,&Xmin,&Xmax,&Ymin,&Ymax);
  x_width=Xmax-Xmin;
  y_width=Ymax-Ymin;

  while (i<=num_of_extrms)
  {
    arr_extremoms[i].x_cor=div_round((arr_extremoms[i].x_cor-
                            Xmin)*GRID,x_width);
    arr_extremoms[i].y_cor=div_round((arr_extremoms[i].y_cor-
                            Ymin)*GRID,y_width);
     i++;
  }

  (*feat0)=div_round(50*x_width,y_width);
}
```

```
#include "main.h"
void print_features (struct FEATURES *feat,char filename[])
{
  FILE *fp;
  int i;

  fp=fopen (filename,"ab+");

  /*fprintf (fp,"name of symbol =%s\n",(*feat).IDfeat);
  fprintf (fp,"size of symbol =%d\n",(*feat).feat0);
  fprintf (fp,"number of points =%d\n",(*feat).feat1);
  fprintf (fp,"number of str =%d\n",(*feat).feat2);
  fprintf (fp,"number of spaces =%d\n",(*feat).feat3);
  fprintf (fp,"number of diff. levels in X for end of
            space=%d\n",(*feat).feat4);
  fprintf (fp,"number of diff. levels in Y for end of
            space=%d\n",(*feat).feat5);
  fprintf (fp,"number of spaces for q<4 =%d\n",(*feat).feat6);
  fprintf (fp,"number of spaces for q>=4 =%d\n",(*feat).feat8);
  fprintf (fp,"number of spaces ro<4
                    =%d\n",(*feat).feat73);
  fprintf (fp,"number of spaces ro>=4
                    =%d\n",(*feat).feat75);

  fprintf (fp,"number of linear strokes
                 =%d\n",(*feat).feat10);
  fprintf (fp,"number of linear strokes where dir<4
                 =%d\n",(*feat).feat69);
  fprintf (fp,"number of linear strokes where dir>=4
                 =%d\n",(*feat).feat71);

  fprintf (fp,"number of strokes with pos.rot
                 =%d\n",(*feat).feat11);
  fprintf (fp,"number of strokes with neg.rot
                 =%d\n",(*feat).feat12);

  fprintf (fp,"number of strokes ro<4
                 =%d\n",(*feat).feat13);
  fprintf (fp,"number of strokes ro>=4
                 =%d\n",(*feat).feat15);

  fprintf (fp,"number of Ext in X neg. rot.
                 =%d\n",(*feat).feat17);
  fprintf (fp,"diff. levels in X for minX   =%d\n",(*feat).feat18);
  fprintf (fp,"diff. levels in Y for minX   =%d\n",(*feat).feat19);

  fprintf (fp,"number of Ext in X pos. rot. =%d\n",(*feat).feat20);
  fprintf (fp,"diff. levels in X for maxX   =%d\n",(*feat).feat21);
  fprintf (fp,"diff. levels in Y for maxX   =%d\n",(*feat).feat22);

  fprintf (fp,"number of Ext in Y neg. rot. =%d\n",(*feat).feat29);
  fprintf (fp,"diff. levels in X for minY   =%d\n",(*feat).feat30);
  fprintf (fp,"diff. levels in Y for minY   =%d\n",(*feat).feat31);

  fprintf (fp,"number of Ext in Y pos. rot. =%d\n",(*feat).feat32);
```

```
    fprintf (fp,"diff. levels in X for maxY    =%d\n",(*feat).feat33);
    fprintf (fp,"diff. levels in Y for maxY    =%d\n",(*feat).feat34);

    fprintf (fp,"number of ext XY for neg. rot=%d\n",(*feat).feat41);
    fprintf (fp,"different levels in X for ext XY-neg.rot
                                               =%d\n",(*feat).feat42);
    fprintf (fp,"different levels in Y for ext XY-neg.rot
                                               =%d\n",(*feat).feat43);

    fprintf (fp,"number of ext XY for pos. rot
                                               =%d\n",(*feat).feat44);
    fprintf (fp,"different levels in X for ext XY-pos.rot
                                               =%d\n",(*feat).feat45);
    fprintf (fp,"different levels in Y for ext XY-pos.rot
                                               =%d\n",(*feat).feat46);

    fprintf (fp,"number of arc parts           =%d\n",(*feat).feat65);
    fprintf (fp,"number of circle parts        =%d\n",(*feat).feat66);

    fprintf (fp,"number of different X-levels =%d\n",(*feat).feat67);

    fprintf (fp,"number of different Y-levels =%d\n",(*feat).feat68);


    fprintf (fp,"sum of square weights
                                         =%d\n",(*feat).sum_sqr_wgt);*/

    fprintf (fp,"\n%s\n",(*feat).IDfeat);
    for (i=0;i<NUM_FEAT;i++)
        {
     fprintf (fp,"%3d",(*feat).feat[i]);
     if ((i==20) || (i==40)) fprintf (fp,"\n");
        }

    fclose (fp);
}
```

```
#include "main.h"
int transform(char infile[30],struct FEATURES *feat)
{
   struct POINT arr_cor[MAX_NUM_OF_POINTS]; /* array of coordinates */
   struct TABLE_EXTREMOMS arr_extremoms[MAX_NUM_OF_EXTREMOMS];
      /* array of special points (i.e. extremums , changing of pen
         position */
   struct TABLE_EXTREMOMS arr_exc_extrms[MAX_NUM_OF_EXTREMOMS];
        /* array of excluded extremums , it is excluded from the array
           of extremums by excluding close points */

   int feat0;
   int num_of_points,num_of_extrms,exc_num_extrms;
/****************************************************************
  Reading a file and calculate the number of points for the symbol,and
  stores the points read in the array : arr_cor .
****************************************************************/
  num_of_points=read_points (arr_cor,infile);
  if (num_of_points==0) return (0);

 /****************************************************************
  Calculates the extremums from the array of coordinates , and stores
  it in the array : arr_extremums , the function returns the number of
  extremums it calculated , and it is stored in num_of_extrms.
 ***************************************************************/
  num_of_extrms=calc_extremoms (arr_extremoms,arr_cor,num_of_points);

 /*************************************************************** *
  Replaces the actual coordinates of the extremums in arr_extremums
  with the levels of these coordinates by doing quantization. (there
  are 16 levels)

****************************************************************/
  conv2levels (arr_extremoms,num_of_extrms,&feat0);

/*************************************************************** *
Excludes close points in the extremums array and stores the remaining
extremums in arr_exc_extrms , it returns the number of extremums left
after excluding.
*************************************************************** */
exc_num_extrms=analyse_extremoms (arr_cor,arr_extremoms,
  arr_exc_extrms,num_of_extrms);


/*************************************************************** *
calculates the features of the symbol , and stores it in the structure
variable feat.

*************************************************************** */
calc_features
 (&(*feat),num_of_points,arr_exc_extrms,exc_num_extrms,feat0);

  return (num_of_points);
}
```

```
#include "main.h"
/*****************************************************************
 a function which recieves as input the features of a symbol , and
 maximum  and minimum features , and returns the estimation of the
 probability that the entered symbol is the symbol with the min and max
 features.
*****************************************************************/
int rec (struct FEATURES feat,struct FEATURES min_feat,
         struct FEATURES max_feat)
{
   int D;
   int theta;
   int W;
   unsigned long temp=0;
   unsigned long sum_w=0;
   int temp1;
   int N=0;
   int Rv;
   int i;

   for (i=0;i<NUM_FEAT;i++)
   {
     if ((feat.feat[i]<=max_feat.feat[i]) &&
         (feat.feat[i]>=min_feat.feat[i]))
        theta=0;
        else {theta=1;N++;}
     if ((min_feat.feat[i]+max_feat.feat[i])==0)
        {
           W=0;
           D=32*feat.feat[i]-16;
        }
     else
      {
    D=abs(div_round(32*feat.feat[i],min_feat.feat[i]+
                                    max_feat.feat[i])-16);
    W=div_round(32*min_feat.feat[i],min_feat.feat[i]+max_feat.feat[i]);
      }
     if ((W==16) && (theta==1)) theta=8;
     temp+=(D*theta);
     sum_w+=W;
   }

   temp1 = (int)100*temp/sum_w;
   Rv = div_round((NUM_FEAT-N)*(100-temp1),NUM_FEAT);

   return max (0,Rv);
}
```

APPENDIX D

```
/* Default values    */
#define N_POINT          500    // Number of points
#define DIMENSION        3      // Number of dimentions
#define N_CENTER         4      // Number of centroids
```

```
#include "start.h"
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
int prepare (int xc[][],int *m1,int *m2,int *m3,int *m4);
void select_beg (int cntr[][],int m1,int m2,int m3,int m4);
float distance (int xc[],int cntr[]);
float sum_dist (int cntr[][],int x[]);


/*        Main program for clustering          */

void main ()
{
  int x[N_POINT][DIMENSION],i1,i2,i3,num_points,i4;
  int center[N_CENTER][DIMENSION],prev_cntr[DIMENSION],
           new_cntr[DIMENS ON];
  int errorcode;
  int max_x,max_y,min_x,min_y;
  long sum_def=0;
  float cur_dist=0,weight[N_POINT],comm_weight,dist=0;

/* Initialization of coordinates of all the points       */

  for (i1=0;i1<N_POINT;i1++)
    for (i2=0;i2<DIMENSION;i2++) {
      x[i1][i2]=0;
    }
  num_points = prepare (x,&min_x,&max_x,&min_y,&max_y) ;
  select_beg(center,min_x,max_x,min_y,max_y);


/* Calculating for every centroid it's new position according
   it's previos position and it's distance from all the points.
                                                             */

  for (i1=0;i1<N_CENTER;i1++) {
    for (i2=0;i2<DIMENSION;i2++) {
      new_cntr[i2] = 0;
      prev_cntr[i2]=center[i1][i2];
    }

/* Calculating the membership value of each point to the current
   centroid                                                  */

    for (i3=0,comm_weight=0;i3<num_points;i3++) {
      if ((cur_dist=distance (&x[i3][0],&center[i1][0]))<0.0001)
        weight[i3]=1.0;
      else
        weight[i3]=(1/cur_dist)/sum_dist(center,&x[i3][0]);
      comm_weight += weight[i3]*weight[i3] ;
      for (i4=0;i4<DIMENSION;i4++)
          new_cntr[i4]+= weight[i3]*weight[i3]*x[i3][i4];
    }
    /* Calculation of the new position of the centroid              */

       for (i4=0;i4<DIMENSION;i4++)  {
         new_cntr[i4]/=comm_weight;
         center[i1][i4] = new_cntr[i4] ;
       }


    /* Checks the movement of the current centroid and stops the
       iterations if it is small (less then 3).                      */

        if (i1==N_CENTER-1) {
         if ((dist+=distance(prev_cntr,&center[i1][0]))<3)
         break;
         dist = 0;
         i1=-1;
       }
       else {
         dist+= distance(prev_cntr,&center[i1][0]);
       }
     }


    /* Printing the points and the centroids                      */

    for (i1=0;i1<num_points;i1++) {
      printf ("\n Point # %d  : ",i1);
      for (i2=0;i2<DIMENSION;i2++)
        printf (" %5d ",x[i1][i2]);
    }
    printf ("\n\n\n\n");
    for (i1=0;i1<N_CENTER;i1++) {
      printf ("\n CENTER # %d  : ",i1);
      for (i2=0;i2<DIMENSION;i2++)
        printf (" %5d ",center[i1][i2]);

    }
    getch ();
  }
```

```
#include "start.h"
/* This procedure selects the initial set of centroids    */

void select_beg(int cntr[N_CENTER][DIMENSION],int min_x,
                int max_x,int min_y,int max_y)
{
   cntr[0][0]=-100;
   cntr[0][1]=-100;
   cntr[0][2]=100;
   cntr[1][0]=200;
   cntr[1][1]=0;
   cntr[1][2]=300;
   cntr[2][0]=0;
   cntr[2][1]=-150;
   cntr[2][2]=400;
   cntr[3][0]=400;
   cntr[3][1]=400;
   cntr[3][1]=50;
/*   cntr[0][4]=0;
   cntr[1][0]=2*max_x;
   cntr[1][1]=2*min_y;
   cntr[1][2]=200;
   cntr[1][3]=100;
   cntr[1][4]=10;
   cntr[2][0]=max_x;
   cntr[2][1]=max_y;
   cntr[2][2]=100;
   cntr[2][3]=100;
   cntr[2][4]=20;
   cntr[3][0]=min_x;
   cntr[3][1]=max_y;
   cntr[3][2]=300;
   cntr[3][3]=100;
   cntr[3][4]=30;
   cntr[4][0]=min_x/3;
   cntr[4][1]=min_y/3;
   cntr[4][2]=200;
   cntr[4][3]=100;
   cntr[4][4]=0;
   cntr[5][0]=max_x/3;
   cntr[5][1]=min_y/3;
   cntr[5][2]=200;
   cntr[5][3]=100;
   cntr[5][4]=10;
   cntr[6][0]=max_x/3;
   cntr[6][1]=max_y/3;
   cntr[6][2]=100;
   cntr[6][3]=100;
   cntr[6][4]=20;
   cntr[7][0]=min_x/3;
   cntr[7][1]=max_y/3;
   cntr[7][2]=300;
   cntr[7][3]=100;
   cntr[7][4]=30;
   cntr[8][0]=min_x*2/3;
```

```
    cntr[8][1]=min_y*2/3;
    cntr[8][2]=200;
    cntr[8][3]=100;
    cntr[8][4]=0;
    cntr[9][0]=max_x*2/3;
    cntr[9][1]=min_y*2/3;
    cntr[9][2]=200;
    cntr[9][3]=100;
    cntr[9][4]=10;
    cntr[10][0]=max_x*2/3;
    cntr[10][1]=max_y*2/3;
    cntr[10][2]=100;
    cntr[10][3]=100;
    cntr[10][4]=20;
    cntr[11][0]=min_x*2/3;
    cntr[11][1]=max_y*2/3;
    cntr[11][2]=300;
    cntr[11][3]=100;
    cntr[11][4]=30;   */
}
```

```
#include "start.h"
#include <time.h>
#include <stdlib.h>
#include <stdio.h>


/* This procedure gets the points from file or from other input
   device and calculates the maximums and minimums for each
   dimention (two dimentions in this example                    */

int prepare(int xc[N_POINT][DIMENSION],int *min_x,int *max_x,
            int *min_y,int *max_y)
{
  int count1,count2;
  FILE * fp;
  *max_x = -10000;
  *max_y = -10000;
  *min_x =  10000;
  *min_y =  10000;

  fp=fopen("strokes.stc","r");
  count1 = 0;
  while ( fscanf (fp,"%6d%6d%6d%6d%6d%6d\n",&count2,
    &xc[count1][0],&xc[count1][1],&xc[count1][2],
    &xc[count1][3],&xc[count1][4]) > 0) {
  /*  xc[count1][0]+=100;
    xc[count1][1]+=100;*/
    *max_x = max(*max_x,xc[count1][0]);
    *max_y = max(*max_y,xc[count1][1]);
    *min_x = min(*min_x,xc[count1][0]);
    *min_y = min(*min_y,xc[count1][1]);
    count1++;
  }
  fclose (fp) ;
  return count1;
}
```

```
#include <math.h>
#include "start.h"

/* This function gets two points and returns the sqr of the
   distance between this points                                   */

float distance (int point1[DIMENSION],int point2[DIMENSION])
{
  int i_c;
  float sum;
  for (i_c=0,sum=0.0;i_c<DIMENSION;i_c++)
    sum+=((float)(point1[i_c]-point2[i_c]))*((float)(point1[i_c]-
                      point2[i_c]));
  return  sum ;
}
```

```
#include "start.h"
float distance(int pnt[],int cntr[]);

/* This function gets a point and the positions of all centroids
   and returns the sum of sqr distances between this point and
   all the centroids */

float sum_dist (int cntr[N_CENTER][DIMENSION],
                int point[DIMENSION])
{
  int ic_1,ic_2;
  float cur_dist;
  float sum_cur=0;
  for (ic_1=0;ic_1<N_CENTER;ic_1++) {
    cur_dist=distance(point,&cntr[ic_1][0]);
    if (cur_dist <.0001)
    cur_dist = 10000.;
    sum_cur += 1./cur_dist;
  }
  return sum_cur;
}
```

CLAIMS

1. Apparatus for reading handwriting comprising:
a sensor for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written; and
circuitry, which is configured for the individual, for providing a non-individual dependent output indicating the symbols being written in response to the sensed features.

2. Apparatus according to claim 1 and wherein said sensor and said circuitry are contained in a hand-held housing.

3. Apparatus according to claim 1 and also comprising a transmitter for wireless communication with a computer to which said transmitter inputs symbol data.

4. Apparatus according to claim 1 and wherein said non-individual dependent output is suitable for communication with the keyboard input of a computer.

5. Apparatus for reading handwriting comprising:
personalized hand-held apparatus for sensing acceleration during handwriting and providing an output indication of handwriting content in a non-personalized form.

6. Apparatus according to claim 5 and also comprising a transmitter for wireless communication with a computer to which said transmitter inputs said output indication.

7. Apparatus according to claim 6 and wherein said output indication is suitable for communication

with the keyboard input of a computer.

8. Apparatus for reading handwriting comprising:
wireless hand-held apparatus for sensing handwriting and providing an output indication of the contents thereof.

9. Apparatus according to claim 8 and wherein said hand-held apparatus senses acceleration during handwriting.

10. Apparatus according to claim 8 and wherein said apparatus is trainable.

11. Apparatus for reading handwriting comprising:

personally trainable hand-held apparatus for sensing motion during handwriting and providing an output indication of handwriting content.

12. Apparatus according to claim 11 and wherein said hand-held apparatus senses acceleration.
13. Apparatus according to claim 11 and also comprising a transmitter for wireless communication with a computer to which said transmitter inputs symbol data.

14. Apparatus according to claim 11 and wherein said non-output indication is suitable for communication with the keyboard input of a computer.

15. Apparatus for reading handwriting in real time comprising:
a hand held housing;
a motion sensor disposed in said housing;
a plurality of parallel recognizers disposed within said hosing and receiving signals from said motion sensor for sensing a plurality of handwriting

characteristics; and

symbol recognizing means disposed in said housing receiving the outputs of said plurality of parallel recognizers for providing an indication of a handwritten signal.

16. Apparatus according to claim 15 and wherein said motion sensor is operative to sense acceleration.

17. Apparatus according to claim 15 and also comprising a transmitter for wireless communication with a computer to which said transmitter inputs symbol data.

18. Apparatus according to claim 1 and wherein said indication is suitable for communication with the keyboard input of a computer.

19. Apparatus for reading handwriting comprising:

hand-held apparatus for sensing motion during handwriting and providing an output indication of handwriting content in a form corresponding to that of a conventional keyboard.

20. Apparatus for reading handwriting comprising:

hand-held apparatus for sensing motion during handwriting and providing an output indication of handwriting content in a RS-232 compatible form.

21. Audio-visual apparatus including apparatus for providing a human sensible output including information in at least one of audio and visual form and having as an input element hand-held apparatus for sensing motion during handwriting.

22. Audio-visual apparatus according to claim 21 and wherein said hand-held apparatus comprises:

a sensor for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written; and

circuitry, which is configured for the individual, for providing a non-individual dependent output indicating the symbols being written in response to the sensed features.

23. Portable information storage and retrieval apparatus including a portable computer memory and output device and having as an input element hand-held apparatus for sensing motion during handwriting.

24. Apparatus according to claim 23 and wherein said hand-held apparatus comprises:

a sensor for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written; and

circuitry, which is configured for the individual, for providing a non-individual dependent output indicating the symbols being written in response to the sensed features.

25. Lock apparatus including locking apparatus responsive to a predetermined electronic input and having as an input element hand-held apparatus for sensing motion during handwriting.

26. Apparatus according to claim 25 and wherein said hand-held apparatus comprises:

a sensor for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written; and

circuitry, which is configured for the individual, for providing a non-individual dependent output indicating the symbols being written in response to the sensed features.

27. Magnetic card activated apparatus including apparatus for reading a magnetic card and having as a verification input element, hand-held apparatus for sensing motion during handwriting.

28. Apparatus according to claim 27 and wherein said hand-held apparatus comprises:

a sensor for sensing features of handwriting of an individual which features are highly characteristic of the individual but which also contain information relating to symbols being written; and

circuitry, which is configured for the individual, for providing a non-individual dependent output indicating the symbols being written in response to the sensed features.

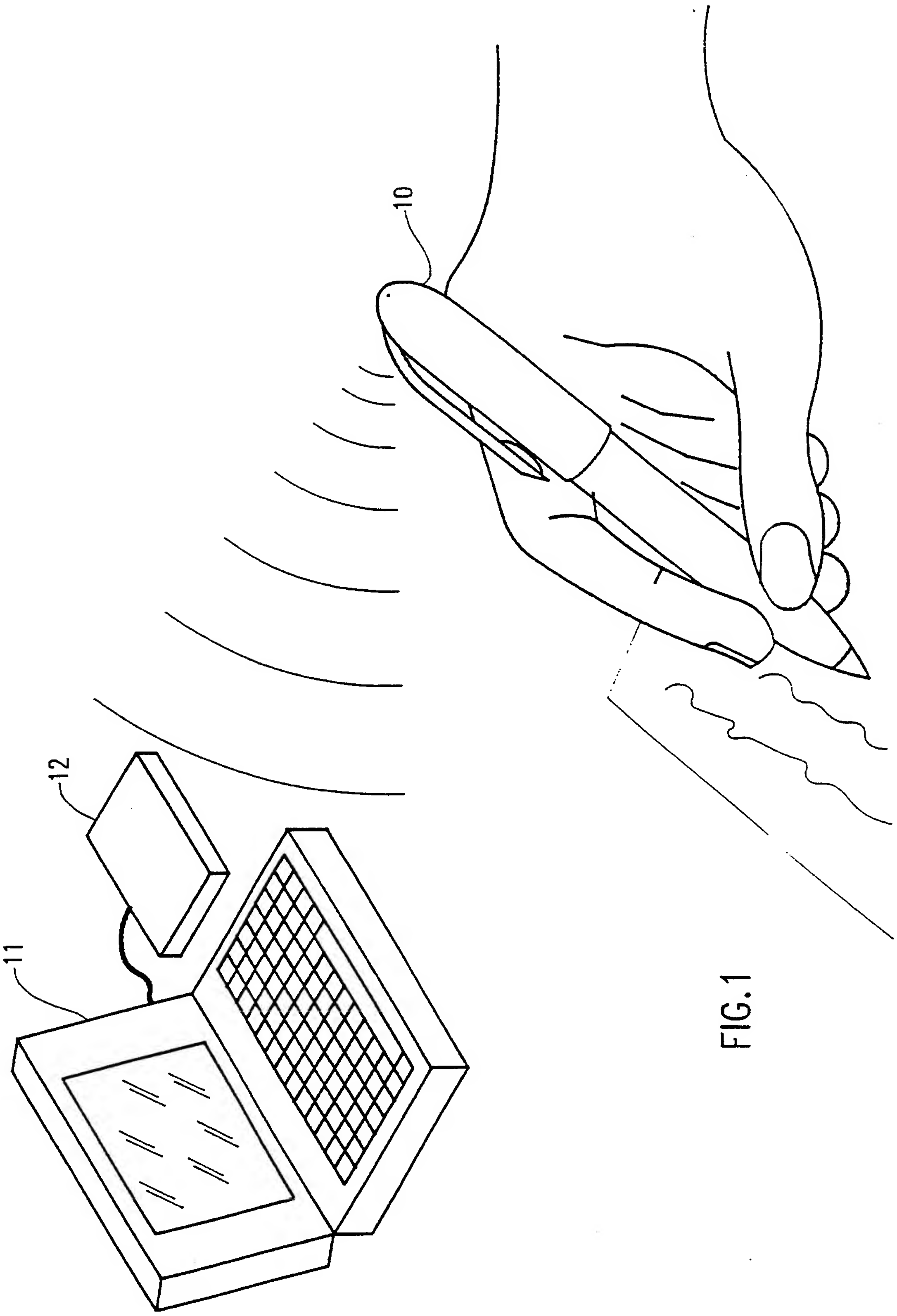
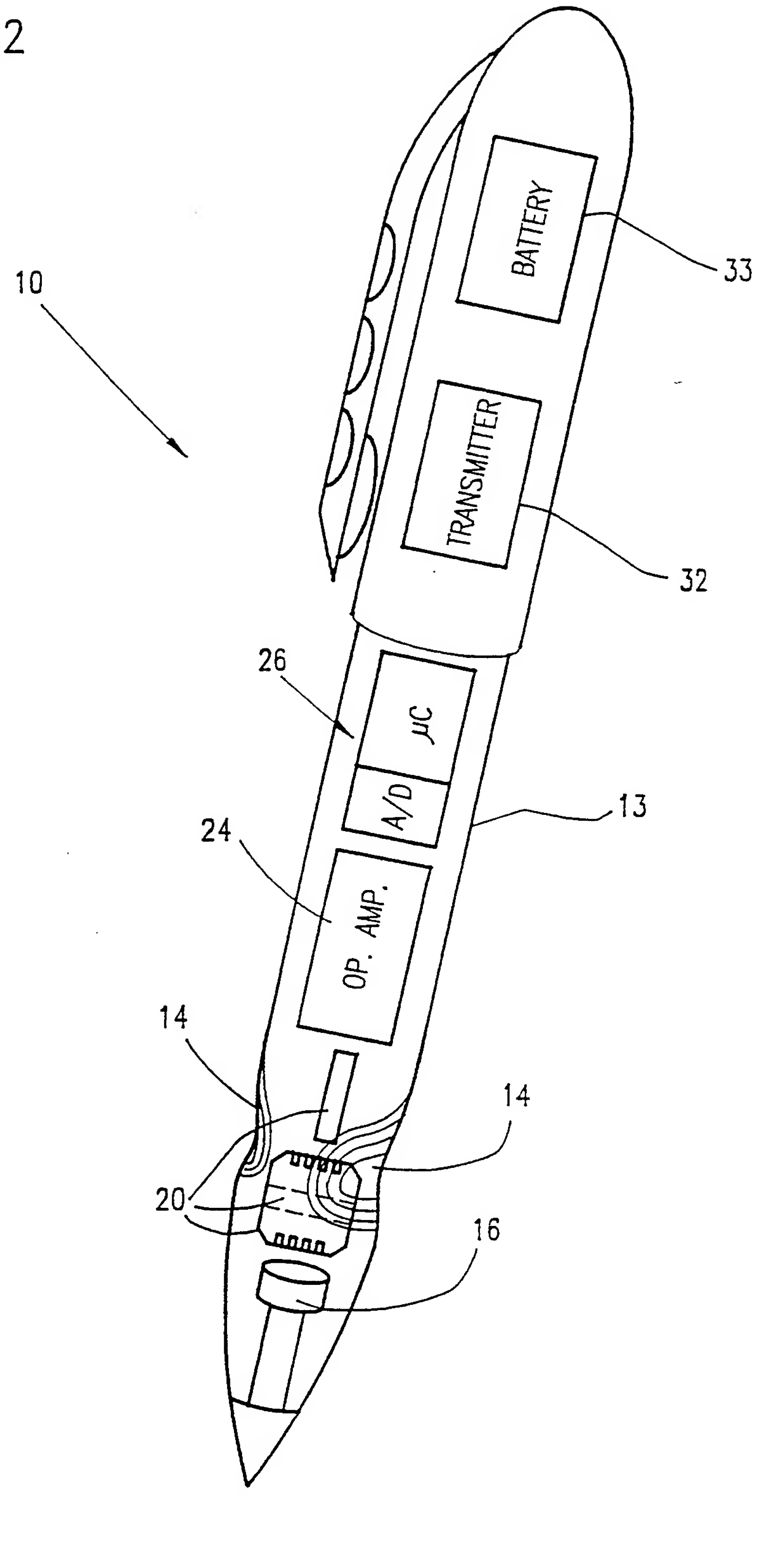


FIG.1

FIG.2



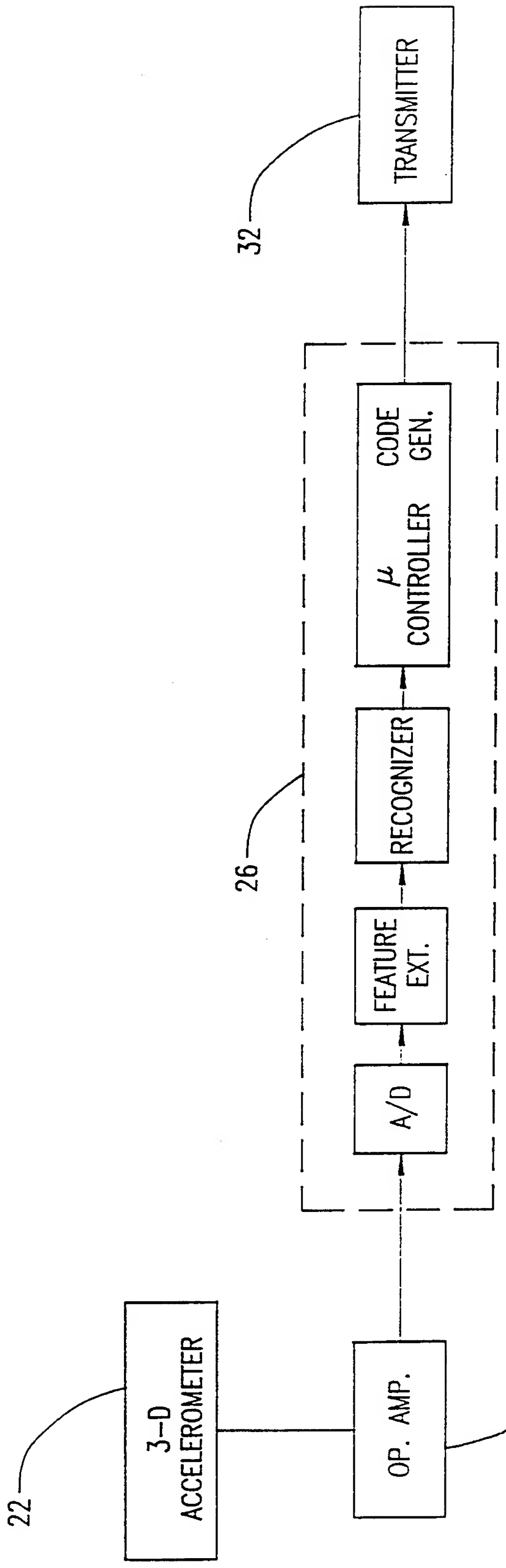
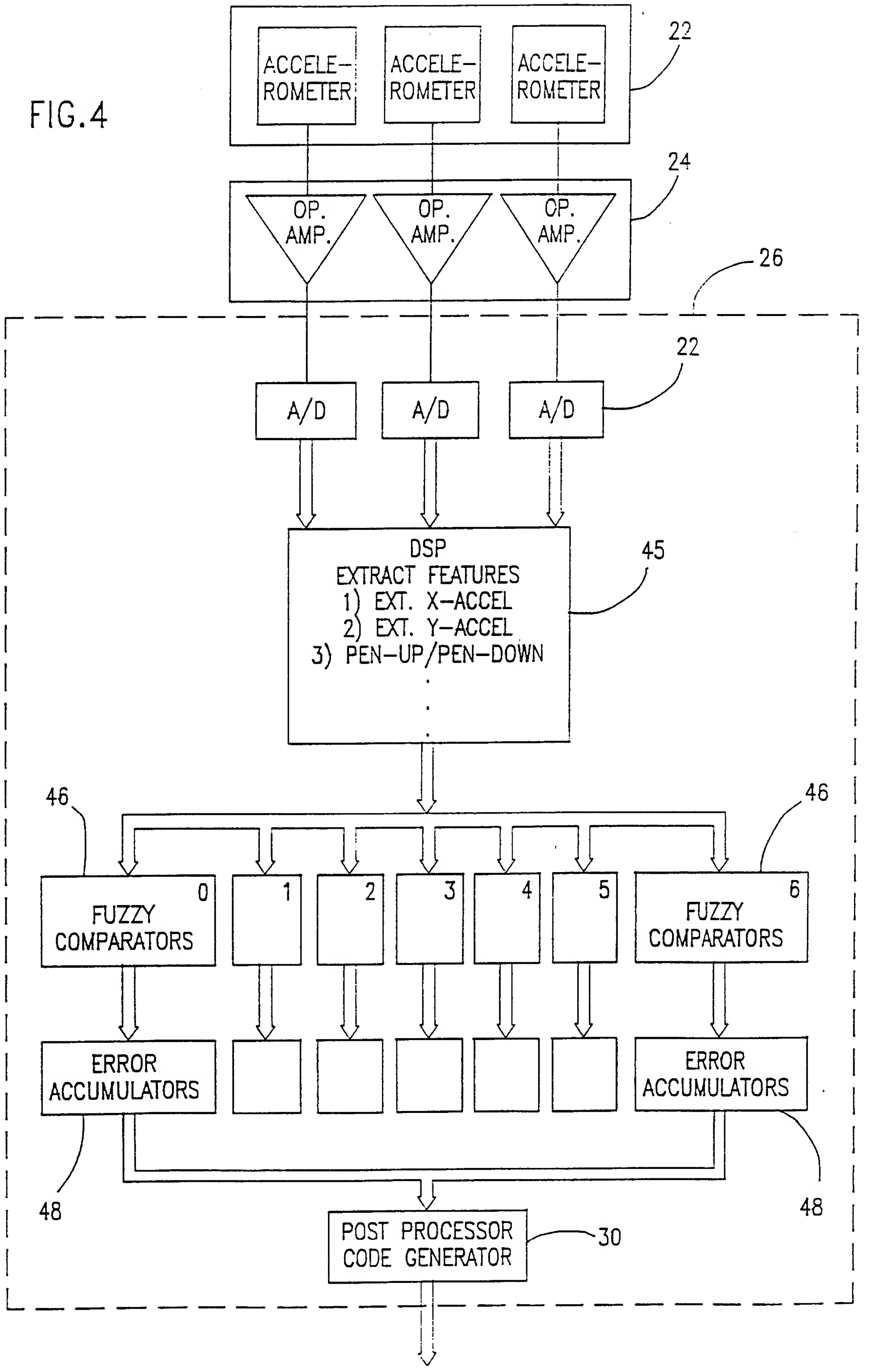

TRANSMITTER
32
CODE GEN.
μ CONTROLLER
RECOGNIZER
26
FEATURE EXT.
A/D
3-D ACCELEROMETER
22
OP. AMP.
24

FIG.3

FIG.4


ACCELE-ROMETER
ACCELE-ROMETER
ACCELE-ROMETER
22
OP. AMP.
OP. AMP.
OP. AMP.
24
26
A/D
A/D
A/D
22
DSP
EXTRACT FEATURES
1) EXT. X-ACCEL
2) EXT. Y-ACCEL
3) PEN-UP/PEN-DOWN
45
46
46
FUZZY COMPARATORS 0
1
2
3
4
5
FUZZY COMPARATORS 6
ERROR ACCUMULATORS
ERROR ACCUMULATORS
48
48
POST PROCESSOR CODE GENERATOR
30

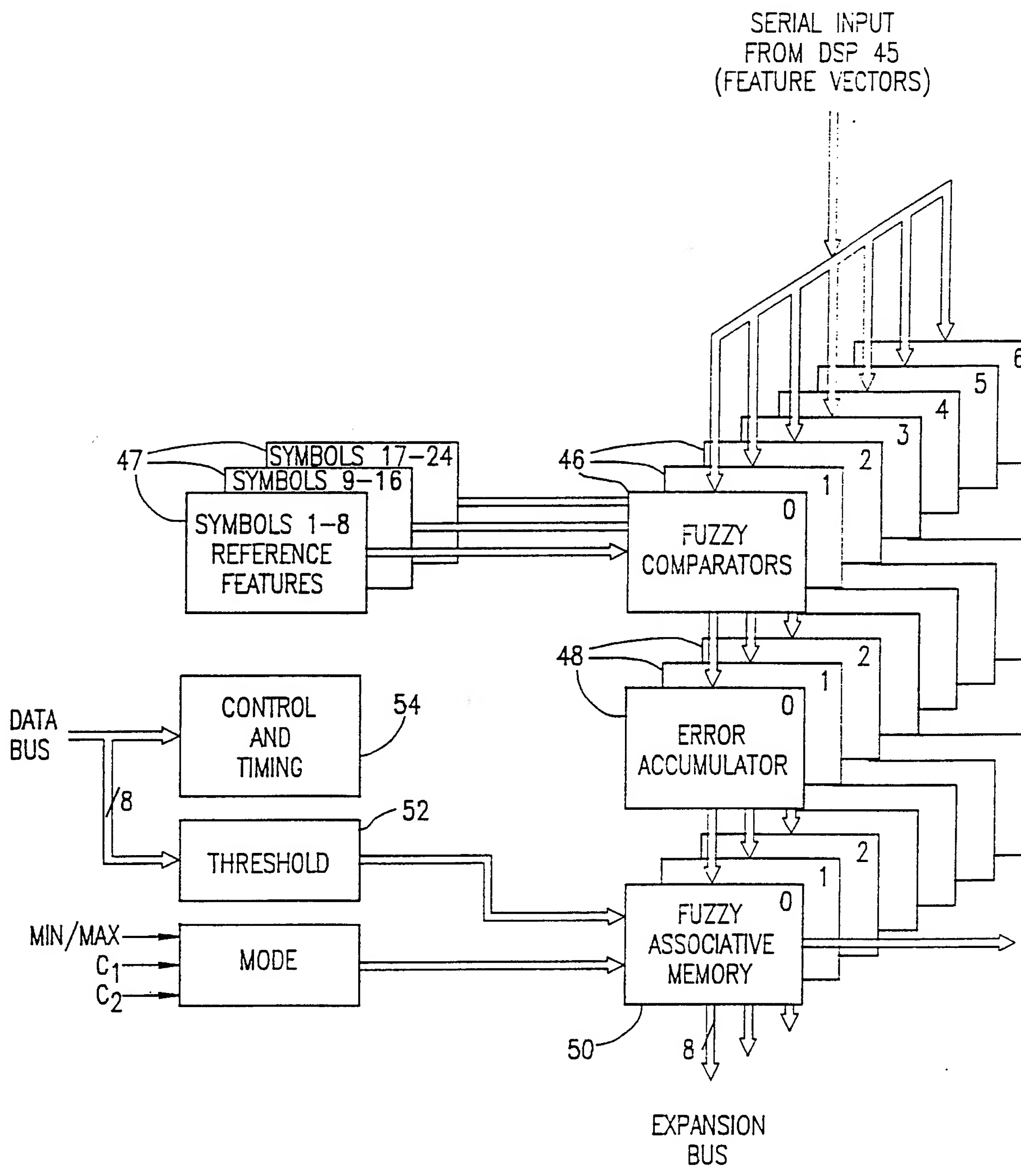

SERIAL INPUT FROM DSP 45 (FEATURE VECTORS)
6
5
4
3
2
1
0
47
SYMBOLS 17-24
SYMBOLS 9-16
SYMBOLS 1-8 REFERENCE FEATURES
46
FUZZY COMPARATORS
48
ERROR ACCUMULATOR
DATA BUS
CONTROL AND TIMING
54
/8
THRESHOLD
52
MIN/MAX
C1
C2
MODE
FUZZY ASSOCIATIVE MEMORY
50
8
EXPANSION BUS

FIG.5

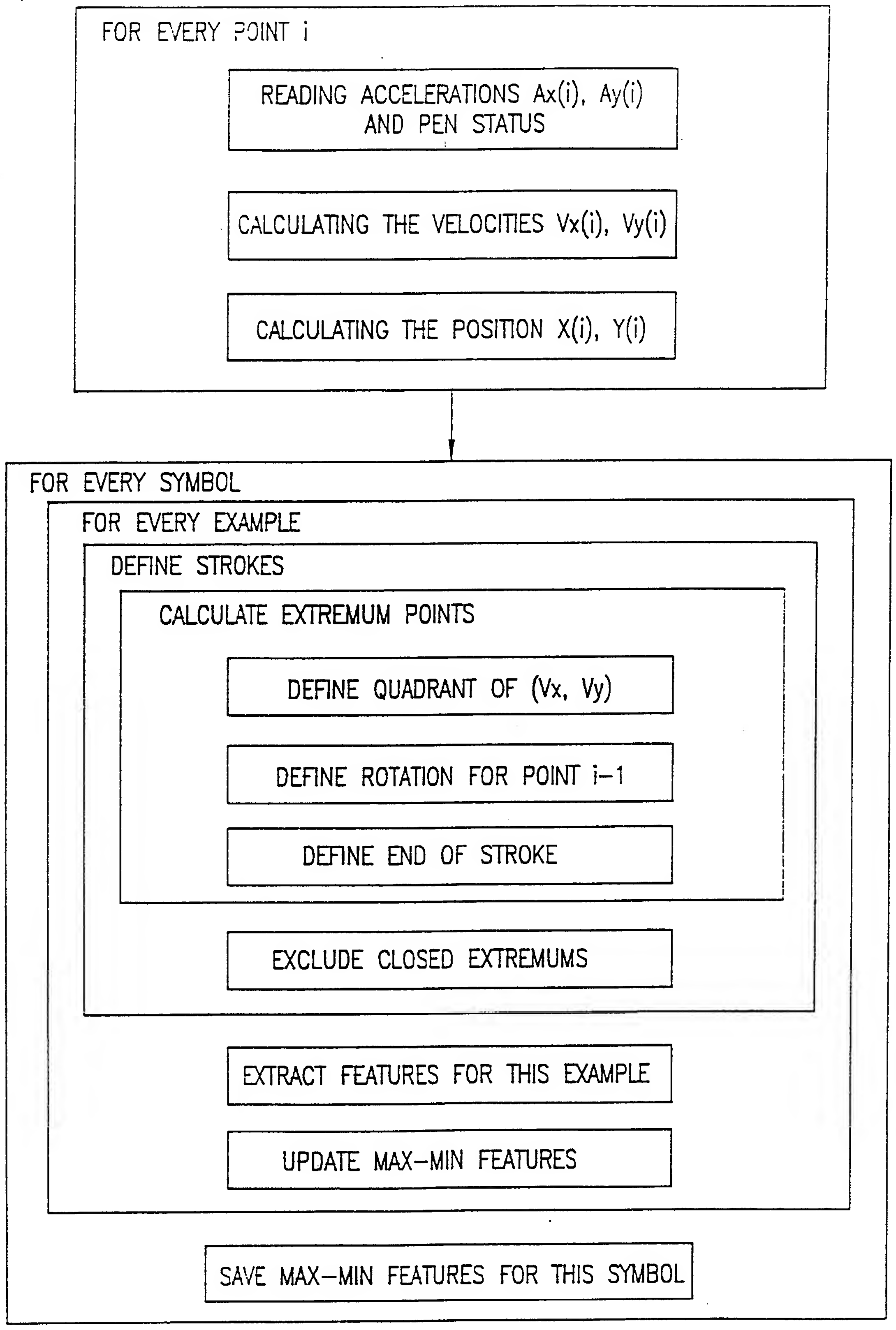

FOR EVERY POINT i
READING ACCELERATIONS Ax(i), Ay(i) AND PEN STATUS
CALCULATING THE VELOCITIES Vx(i), Vy(i)
CALCULATING THE POSITION X(i), Y(i)
FOR EVERY SYMBOL
FOR EVERY EXAMPLE
DEFINE STROKES
CALCULATE EXTREMUM POINTS
DEFINE QUADRANT OF (Vx, Vy)
DEFINE ROTATION FOR POINT i-1
DEFINE END OF STROKE
EXCLUDE CLOSED EXTREMUMS
EXTRACT FEATURES FOR THIS EXAMPLE
UPDATE MAX-MIN FEATURES
SAVE MAX-MIN FEATURES FOR THIS SYMBOL

FIG.6A

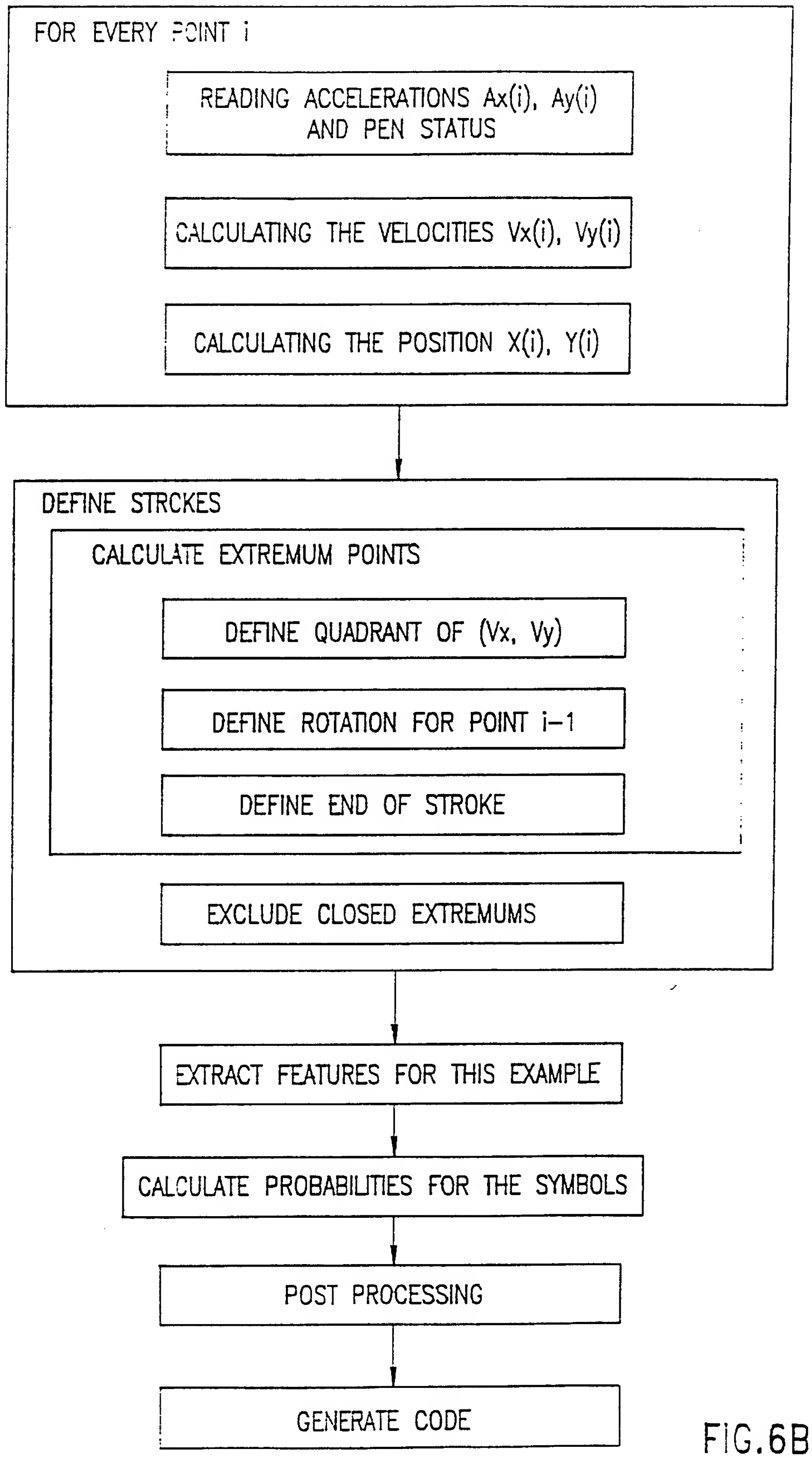

FOR EVERY POINT i
READING ACCELERATIONS Ax(i), Ay(i) AND PEN STATUS
CALCULATING THE VELOCITIES Vx(i), Vy(i)
CALCULATING THE POSITION X(i), Y(i)
DEFINE STROCKES
CALCULATE EXTREMUM POINTS
DEFINE QUADRANT OF (Vx, Vy)
DEFINE ROTATION FOR POINT i-1
DEFINE END OF STROKE
EXCLUDE CLOSED EXTREMUMS
EXTRACT FEATURES FOR THIS EXAMPLE
CALCULATE PROBABILITIES FOR THE SYMBOLS
POST PROCESSING
GENERATE CODE

FIG.6B

INTERNATIONAL SEARCH REPORT

PCT/US92/08703

A. CLASSIFICATION OF SUBJECT MATTER

IPC(5) :G06K 9/00
US CL :382/3
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 382/3, 13 58, 59

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	US, A, 5,027,414 (HILTON) 25 June 1991, see Figures 1-3 and refer to column 3, line 20 through column 6, line 2.	1-14, 18-26 15, 27, 28
X	US, A, 4,495,644 (PARKS ET AL.) 22 January 1985, see Figure 2 and column 15, line 50 through column 18, line 2.	5, 11, 12, 14, 19-26
X Y	US, A, 5,107,541 (HILTON) 21 April 1992, see Figure 1-8 and refer to column 4, line 60 through column 10, line 57.	1-14, 18-26 15, 27, 28

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 18 NOVEMBER 1992	Date of mailing of the international search report 22 JAN 1993
Name and mailing address of the ISA/ Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer [illegible] J.C. COUSO
Facsimile No. NOT APPLICABLE	Telephone No. (703) 305-4774